This chapter deals with the literature review and existing MASs and is organized as follows:

- Section 2.1 describes the literature regarding load balancing. It includes static vs. dynamic, homogeneous vs. heterogeneous, centralized vs. distributed load balancing literature. In case of distributed load balancing, the literature related to the issues like fault tolerance, resource management, route and service discovery, security and intrusion detection is also given in Section 2.1

- Section 2.2 deals with description of some existing MASs

- Section 2.3 summarizes the chapter

### 2.1 Load Balancing

In general, load balancing provides an ability to avoid the situation where some resources of the systems are overloaded while others remain idle or under loaded. It is well understood that excessively overloading a portion of resources substantially reduces the overall performance of the systems. This section presents a summary of the work related to load balancing techniques. Specifically, we describe the important features in a wide range of load balancing approaches.
2.1.1 Static vs. Dynamic

Existing load balancing approaches are classified into two broad categories - Static and Dynamic. In each category, there are further classifications to associate specific attributes. Figure 2.1 shows a simplified taxonomy of load balancing schemes most relevant to this research. A large amount of work has been done for static load-balancing schemes that do not rely on the current state of hosts [4]. Tantawi and Towsley [5] have exploited optimal static load balancing for a single job class in star and bus network configurations. Based on this study, Kim and Kameda [6] have proposed two static load balancing algorithms, which are quite effective to improve system performance. Chronopoulos et al. [7] have developed a static load balancing scheme for Computational Fluid Dynamics simulations on a network of heterogeneous workstations. In contrast, the dynamic load balancing approaches provides an ability to improve the quality of load distribution at run time at a reasonable cost of communication and processing overhead. McCann et al. [8] have studied a dynamic scheduling strategy, which is aware of resource requirements of submitted tasks. Their design is based on a centralized manager that handles scheduling on each processor, but the approach is not very scalable. Condor [9] has been developed to harvest the idle cycles of a cluster of computers by distributing batch jobs to idle workstations. One of the ultimate goal of Condor is the guarantee that workstations immediately become available for their owners when the owners are about to access the machines. This goal is approached by detecting an owner’s activity at an idle machine, and migrating background jobs to other idle machines when the owner begins accessing his/her machine. For reliability purposes, Condor periodically makes checkpoints on tasks, thereby making it possible to restore and resume jobs in presence of
software and hardware failures. In addition, Condor offers a flexible mechanism where each machine’s owner specify conditions under which the machine is considered idle. Watts and Taylor [10] have examined a cohesive and practical dynamic load balancing framework, which is scalable, portable and easy to use. The approach aims to improve the existing strategies by incorporating a new diffusion algorithm to offer a good tradeoff between total work transfer and run time. In addition, the load balancing approach takes advantage of a task selection mechanism, allowing the task size and the communication cost to guide the task movement. Harchol-Balter and Downey [11] have proposed a CPU-based dynamic load balancing policy that is effective in improving CPU utilization. Zhang et al. [12] have focused on dynamic load sharing policies by considering both CPU and memory services among the hosts. The experimental results show that their policies not only improve performance of memory-intensive jobs, but also maintain the same load sharing quality for CPU-intensive jobs. The work presented only addresses the dynamic load balancing problem.

2.1.2 Homogeneous vs. Heterogeneous

At another level, load balancing approaches are broadly classified into homogenous and heterogeneous [Figure 2.1]. The focus of homogeneous load balancing schemes is to improve the performance of homogeneous parallel and distributed systems. Colajanni et al. [13] have addressed the issue related to dynamic load balancing in context of distributed homogeneous Web servers. The results show that their strategies are robust and effective in balancing load on web servers.
On the other hand, heterogeneous load balancing approaches attempt to boost the performance of heterogeneous clusters, which comprise a variety of hosts with different performance characteristics in terms of computing power, memory capacity, and disk speed. Some dynamic scheduling policies for heterogeneous parallel systems can be found in [14]. To dynamically balance the computational loads in a heterogeneous cluster environment, Cap and Strumpen [15] have explored heterogeneous task partitioning and load balancing schemes.

### 2.1.3 Centralized vs. Distributed

This section explores load balancing, fault tolerance, resource discovery, service discovery, and security with intrusion detection on centralized and distributed control type of networks.
In centralized approach, the global state information is collected or estimated at a single host (server) which makes request (task) distribution decisions based on the collected information. This approach may impose fewer overheads for maintaining the state informations, but has lower reliability. Failure of the central server makes load sharing inoperable.

A variety of load balancing schemes have been proposed [16, 17, 18, 19] for cluster. In particular, dynamic load balancing schemes have been extensively investigated, primarily focusing on CPU [19, 20], memory [18, 21], network [16, 22, 23], or a combination of CPU and memory [24, 25] resources. Two main approaches to balance load on cluster disk I/O resources can be found in [25, 26]. Zhang et al. [25] have proposed three I/O-aware scheduling schemes that are aware of the job’s spatial preferences. Lee et al. [26] have studied two file assignments algorithms to balance load across all the disks on a system, thereby making it possible to improve overall system performance by fully utilizing available hard drives.

Load balancing is studied on a Grid which has emerged as an important new field. It is distinguished from conventional distributed computing with its focus on large-scale resource sharing, innovative applications, and high-performance orientation [26, 27]. The main components of a grid infrastructure are security, resource management services, information services, data management services and load balancing [28]. Various grid services are offered under a grid environment, which is defined as the web service that provides a set of well-defined interfaces and that follows specific conventions [29, 30]. Many services offered by the grid need to access data from a certain source database, such as the Bio Map [31].

The partition of a service task into subtasks and their distribution among available resources is of great concern, because they significantly affect the grid service reliability, cost and profits [32]. A common grid service model
allows agents to represent various grid resources, which are owned by different real world enterprises. The grid task agents buy resources to complete the tasks. Grid resource agents charge the task agents for an amount of resource capacity allocated [33]. In the meantime, the grid task agents charge users who requested the service [34]. The optimal task/resource scheduling problem and its significance can be found in [34, 35]. Some other optimization schemes, proposed for grid are described in [36, 37, 38].

In recent times, fault tolerance on grid has emerged as an attractive computing paradigm for researchers all over the world. There are various approaches to make the Grid fault tolerant [39, 40, 41]. The Globus project [42] provides a heartbeat service to monitor the running processes to detect faults. The application is notified of the failure and is expected to take appropriate recovery action. Legion [43, 44] provides mechanism to support fault tolerance with checkpointing on grid. Other grid systems like Netsolve [45, 46, 47] have their fault detection and failure recovery mechanisms. They provide a single user transparent failure recovery mechanism. FATCOP [48] is a parallel mixed integer program solver that works in an opportunistic computing environment provided by the Condor resource management system, using an implementation of a branch-and-bound algorithm.

For scheduling on grid, there are several systems that have been developed. The most significant attempts can be found in meta-schedulers such as Nimrod-G [49, 50] with software execution environment such as GRADS [51] and task brokers such as Condor-G [52]. The latter is a product of a much more complicated entity that consolidates scheduling policies with workload management systems. Additionally, AppLeS [53] is a scheduling system which primarily focuses on developing scheduling agents for individual applications on production. Other interesting works on scheduling
and meta-scheduling are presented in [54] and [55] where, in the former, the authors present a heuristic scheduling of bag-of-tasks with QoS constraints, while the latter handles the problem of distributed job scheduling in a grid using multiple requests.

One of the solutions to Fault Tolerance [39-48] on grid is achieved using checkpointing by saving the state of the process on some stable storage media [56, 57]. Checkpointing algorithms are classified into three broad categories: (a) synchronous, (b) asynchronous and (c) quasi-synchronous [58]. In asynchronous checkpointing [59, 60] each process takes checkpoints independently. In synchronous checkpointing [56, 57, 61, 62–64, 58, 65, 66, 67], each process synchronizes through system messages before taking checkpoints.

In asynchronous checkpointing [68], processes take checkpoints without any coordination. Quasi-synchronous [58] checkpointing approach is a tradeoff between synchronous and asynchronous checkpointing. Koo and Toueg [62], Spezialetti and Kearns [69], Prakash and Singhal [70] and Mandal and Mukhopadhyaya [63, 64] have proposed methods for handling concurrent initiations of snapshot. Not much is reported in the literature except [63, 71] on checkpointing and recovery using MAs.

Some work on performance evaluation of checkpointing and rollback recovery algorithms have been reported in the literature. Plank and Thomason [72] have calculated the average availability of parallel checkpointing systems and used it in selecting runtime parameters like number of processors and checkpointing interval. These minimize the expected execution time of a running program in the presence of failures. Vaidya [73] proposed a two level distributed recovery scheme to achieve better performance than the traditional recovery schemes. The same algorithm was also analyzed by Panda and Das [74] by taking the probability
of task completion on a system with limited repair as the performance metric. Rao et al. [75] have presented an experimental evaluation of the performance of different message logging protocols during recovery.

In distributed approach, each host in the group communicates with others and makes its own decisions for sending the requests to other hosts in the group, or for obtaining the requests from them. Hosts are not dependent on the centralized control for decision making. There are two types of networks in distributed approach (Figure 2.1): P2P and Ad Hoc. Now we will present the literature related to P2P and Ad Hoc networks.

In the last several years there is an emergence of a class of structured P2P networks ([76], [77], [78], [79]). In such systems, object IDs are uniformly distributed. Several solutions have been proposed to address the load balancing problem [77], [80], [81], [82] in P2P networks. Under the assumption that object IDs are uniformly distributed, the number of objects per host varies within a factor of $o(\log n)$. CAN [76] improves this factor by considering a subset of existing hosts (i.e., a host along with its neighbors) instead of a single host. Chord [77] was the first to propose the notion of virtual servers to improve load balancing. Byers et. al. [81] have proposed the use of the power of two choice paradigm to achieve better load balancing. In this paradigm, each object is hashed to a different ID, and is placed in the least loaded host responsible for the ID. Adler et al. [83] have presented a Dynamic Hash Table (DHT) which provably ensures that, as hosts join the system, the ratio of loads of any two hosts is $O(1)$ with high probability. Karger and Ruhl [84] have proposed algorithm which dynamically balance load among peers without using multiple virtual servers by reassigning lightly loaded hosts to be neighbors of heavily loaded hosts. Douceur and Wattenhofer [85] have proposed algorithms for replica placement in a distributed file system. Triantafillou et al. [86] have studied
the problem of load balancing in the context of content and resource management in P2P networks. The authors consider an unstructured P2P network, in which meta-data is aggregated over a two-level hierarchy. Aggarwal et al. [87] have proposed an offline setting of periodic load balancer, which minimizes both the maximum load and the amount of load moved. Westbrook [88], Andrews et al [89], and Azar’s [90] have presented online algorithms for load balancing.

Mobile Ad Hoc Network (MANET) is a collection of wireless mobile hosts forming a temporary network without the aid of any centralized administration or standard support services regularly available in wide area networks to which the hosts may normally be connected. Numerous routing protocols have been developed for MANET. These protocols are generally categorized as table driven, and on demand routing. Table driven routing protocols [91–95] attempt to maintain consistent, up-to-date routing information on each host by propagating updates throughout the network. On the other hand, on demand routing protocols [96–100] maintain routing information at every host, but create routes only when needed by a source host. The other hybrid routing protocols [101-102] that route packets through cluster heads [103] have been proposed. Cluster heads are special hosts that emulate the behavior of base stations in wireless cellular networks.

Among all proposed MANETs protocols, Dynamic Source Routing (DSR) [98, 99] and Ad Hoc On demand Distance Vector (AODV) [100] are the most prominent, and have been submitted to the Internet Engineering Task Force (IETF) MANET working group [104] as candidates for standardization. The packet delay for both AODV and DSR increases as the host mobility is reduced [105]. Source initiated on demand routing protocols discover a route only when needed. Some of the prominent protocols in this category are DSR [98] and Neighborhood Aware Source Routing (NSR)
[106]. NSR protocol maintains information about 2-hop neighborhood of a host. Alternate Path Routing (APR) [107] provides load balancing by distributing traffic among a set of diverse paths, and studies the impact of route coupling on APR performance. In Dynamic Load Aware Routing (DLAR) [108], hosts use their load information to select a route. Papadimitratos et al. [109] presented a protocol for finding the maximum number of paths between a source and destination with highest reliability. Lou et al. [110] have given another method for finding the maximum number of the most secure paths named Security Protocol for Reliable Data Delivery (SPREAD). Aristotelis et al. [111] explore a multipath routing for frequent topology changes.

Service discovery is an integral part of MANETs. While several service discovery protocols such as Service Location Protocol (SLP) [112] and Universal Plug and Play (UPnP) [113] have been developed, most of them are designed for infrastructure based networks and thus not suitable for MANETs. A service may be a computation, storage, a communication channel to another user, software, or a hardware device needed by another user [114]. There are currently a number of existing protocols for service discovery [113, 115, 116, 117]. Also agent platform used in e-commerce [118] have been designed to facilitate flexible service/agent discovery with an agent community. In order to utilize directories in service discovery protocols in MANETs, Kozat et al. [119] have suggested the similarity between directory formation and maintenance, and the idea of clustering is used in some MANET network routing protocols. Although clustering based routing protocols usually incur high overhead for maintaining the clusters, Chen et al. [120] argue that the efficiency of the whole system, nevertheless, improves by using directories in a service discovery protocol. SLP [112] is a protocol for automatic resource discovery on IP based networks. Another
solution to service discovery is Universal Plug and Play (UPnP) [113], supported by Microsoft. Several solutions have been proposed to resolve the issue of resource discovery. Johnson et al. [121] have suggested the use of the virtual backbone to probe and cache the context information for providing a better performance. Belligemine [122] has suggested a rating system that evaluates context information by placing different weights on different context information. There are several platforms that facilitate service discovery given by Foundation for Intelligent Physical Agent (FIPA). One of the platforms is FIPA-OS [123]. LEAP [124] and Micro FIPA OS [125] have been developed to extend the functionality that allows mobile devices to participate in heterogeneous networks.

Unlike conventional cellular wireless mobile networks that rely on extensive infrastructure to support mobility, MANETs do not need expensive base stations or wired infrastructure. Every mobile host is supposed to be a friendly host and is willing to send messages for others. Intrusion Detection Systems (IDSs), serve as a line of defense for MANETs with high security requirements. MANET IDSs complement and integrate with existing MANET intrusion prevention methods to provide highly survivable networks [126]. Intrusion detection is a security technology that attempts to identify individuals who are trying to break into and misuse a system without authorization and those who have legitimate access to the system but are abusing their privileges [127]. The system protected is used to denote an information system being monitored by IDS. It can be a host or network equipment, such as a server, a firewall, a router, or a corporate network [128]. Relatively few research efforts have been devoted to wireless IDSs. Kachirski et al. [129] have proposed distributed IDS for MANETs based on MA technology. Samfat et al. [130] have proposed an Intrusion Detection Architecture for Mobile Networks (IDAMN). Martí et al. [131] have
presented a routing misbehavior in MANETs. Host misbehaves by agreeing to forward packets and then failing to do so, because it is overloaded, selfish, malicious, or broken. An agent is attached to each mobile host, and each host in the network participates in the intrusion detection and response [132]. Huang et al. [133] give a new data mining method that performs the cross feature analysis to capture the inter feature correlation patterns of MANET. Several alert aggregation and correlation techniques [134-139] have been proposed to facilitate the analysis of intrusions. Debar et al. [134] have proposed aggregation and correlation component. Cuppens et al. [135-136] use Lambda language to specify attack scenarios. Valdes et al. [137] give a probabilistic method that correlates alerts using the attribute similarity among their features. Ning et al. [138] have developed three utilities to facilitate the analysis of large sets of correlated alerts. Morin et al. [139] give a formal data model called \textit{M2D2} in order to make full use of the available information. The effectiveness of the proposed aggregation and correlation algorithms depends heavily on the information provided by the individual IDS. Puttini et al. [140] has worked on Hierarchical IDS architecture according to distributed features of MANETs. Sun et al. [141] give the mechanism for alert aggregation in Ad Hoc network. Patel and Upinder [142] have proposed a framework for intrusion detection using MAs.

2.2 Some Existing Mobile Agent Systems

This section presents different types of MAS- Agent TCL [143], ARA [147], Concordia [145], TACOMA [149] and PMADE [2]. Though these systems differ in their goals, motivations, and implementations, they all (more or less) provide common functionalities that support migration of agents, communication between agents, various programming/interpreted languages, and various forms of security.
2.2.1 Agent TCL/D’Agents

Agent TCL (later renamed D’Agents) is a MAS created at Dartmouth College to address the weaknesses of existing MASs, such as insufficient security mechanisms, support for only specific and complex languages, difficult or nonexistent communication between agents, and inadequate migration facilities [143]. The architecture of Agent TCL is based on the server model of Telescript and supports a modified version of the Tool Command Language (TCL) as its high-level scripting language implementation [143] (support for Java [144]). The architecture of Agent TCL consists of four levels:

The lowest level contains transport mechanism. The next level is the server level that manages local and incoming agents. A server that runs at each machine of the network performs tasks such as:

- Keeping track of agents on local machine
- Accepting and authenticating incoming agents
- Providing a hierarchical namespace for each agent and service
- Allowing agent communication via messages
- Allowing agent migration
- Providing access to a non-volatile storage so that agents can save and restore their internal state as in case of a host failure [143]

The next level is the interpreter level which provides the execution environment for each supported agent language [144]. The last level is the agent level that contains the agents themselves. The agents execute in the interpreters and use the facilities provided by the server to migrate from machine to machine and to communicate with other agents [144]. There are two types of agents- those that move from machine to machine accessing resources, and those that remain on the machine. Their purpose is to provide
specific services not inherently provided by the system (e.g., navigation, high-level communication protocols, and resource management) [144].

Agents migrate using the agent_jump command. The modified TCL language provides this command. The method used in moving agents is determined by the transport mechanisms supported by the server. The agent_jump command captures the internal state of an agent and sends this information to destination machine. The server on the destination machine loads the appropriate interpreter for the agent, restores the migrated agent’s state information into the execution environment, and resumes the agent’s execution at the statement immediately after the agent_jump [143]. The agent is now on the destination machine and can interact with that machine’s resources without any further network communication [144]. Agent TCL also provides the simple commands like agent_meet, agent_accept, agent_send, and agent_receive. Agent_meet and agent_accept are used to establish a direct connection between agents [144].

Security in Agent TCL is provided in various capacities. To protect migrating agents and to provide authentication (e.g., to verify the identity of an agent’s owner), Agent TCL uses Pretty Good Privacy (PGP) for its digital signatures and encryption [144]. To protect resources, a resource manager assigns each agent a set of access permissions [144]. So, when an agent tries to access a resource, the request is sent to resource manager that checks the agent’s access permission with the resource. If the agent does not have the proper permission, it is denied to access the resource. Agent TCL has been used in both information management and information retrieval applications [143].
2.2.2 Concordia

It is used for development and management of network-efficient MA applications for accessing information anytime, anywhere, and on both wired and wireless device supporting Java [145]. The applications move around the network machines running Concordia to access services such as databases and those provided by other agents. Concordia is a full featured framework developed at Mitsubishi Electric Information Technology Center America’s (MEITCA) Horizon Systems Laboratory [146]. At the highest level, a Concordia system consists of a Java Virtual Machine (JVM), a Concordia Server running on a machine in a network, and MA running in the system. Both the Concordia server and MAs are Java programs. JVM is used for Concordia’s runtime environment [146].

Concordia consists of a set of components that provides services such as communication, security, persistent storage, administration, and so on. The component responsible for agent mobility is the Conduit Server. When an agent wants to initiate its transfer to another machine, it invokes the methods provided by the Conduit Server [145]. The Conduit Server will then suspend the agent and creates a persistent image of it to be transferred [146]. The Conduit Server will inspect the agent’s Itinerary to determine its destination, contact the Conduit Server on the destination machine, and transfer the agent’s image to destination where it is again stored before being acknowledged and then it can resume execution [146].

Concordia’s security model provides two types of protection- protection of agents from being tampered with, and protection of server resources from unauthorized access [145]. To protect resources on each server, Concordia relies on its Security Manager component to manage resource protection. Each agent is assigned an identity that is used when it tries to access the resources. The Security Manager authenticates each agent by verifying its
identity. If the identity matches, then the agent is able to access the resource. Concordia’s resource protection is based on the user of the agent rather than the developer of the agent, as in other systems [145].

2.2.3 ARA
ARA is a platform for the portable and secures execution of MAs in heterogeneous networks. MAs in this sense are programs with an ability to change their host machine during execution while preserving their internal state. This enables them to handle interactions locally which otherwise had to be performed remotely. ARA's specific aim in comparison to similar platforms is to provide full MA functionality while retaining established programming models and languages. Core is the central part of an ARA system, implementing the basic concepts such as agents, allowances, service points, migration etc [147]. The core for reasons of security and portability mediates any access from an application agent to the host system or to another agent. The core treats agent independently using assistance from the language interpreters for language specific tasks. ARA is primarily concerned with system support for general MAs regarding secure and portable execution, and secondarily with application-level features of agents, such as agent cooperation patterns, intelligent behavior, user modeling etc. Mobility is integrated as comfortably and unintrusively as possible with existing programming concepts-algorithms, languages, and programs [148].

Most of the existing platforms do not run the agents on the real machine of processor, memory and operating system, but on some virtual one, usually an interpreter and a run-time system, which both hides the details of the host system architecture as well as confines the actions of the agents to that restricted environment. This is also the approach adopted in ARA. MAs are programmed in some interpreted language and executed within an interpreter.
for this language using a special run-time system for agents, called the core in ARA system [148].

The programming model of ARA consists of agents autonomously moving between and staying at places, where they can use certain services, provided by the host or other agents, to do their job. A place is physically located on some host machine, and may impose specific security restrictions on an agent entering that place in form of a local allowance limiting the agent's resource accesses while staying at that place [147]. Keeping this in mind, agents are programmed much like conventional programs in all other respects, i.e. they work with a file system, user interface and network interface. The system offers a clear interface to adapt interpreters for established programming languages to the core, demonstrated by the adoption of interpreters for such diverse languages such as C/C++ and Tcl. ARA offers full migration of agents, i.e. orthogonal to conventional program execution, which relieves the programmer of all details involved with remote communication and state transfer. ARA agents migrate at any point in their execution, simply by using a special core call named *ara_go* in Ara’s Tcl interface [148]. The security model of Ara is flexible in the domain of protected resources. These resources are dynamically created in the form of places, and that the admission of agents to such a domain, as well as their actual rights at that place, can be controlled in a fine grained manner to individual agents and resources. However, the described architecture [147] is still lacking in the area of structured agent interoperation.

2.2.4 TACOMA

An agent in TACOMA is a piece of code that is installed and executed on a remote computer. Such an agent explicitly migrates to other hosts in the network during execution. The TACOMA project focuses on operating system support for agents and how these agents can be used to solve
problems traditionally addressed by other distributed computing paradigms, e.g. the client/server model. An agent stores code and data for future computations. It is able to carry this information around when it migrates, and later retrieve it. Also, agents are allowed to leave data behind at hosts or share data with other agents. A folder represents this type of information in the TACOMA system. TACOMA agents store data in folders. A subset of folders are identified with individual hosts and collected in the file cabinets managed by the hosts, the remaining folders comprise a briefcase that is moved from host to host along with the computation. Folders are organized in briefcases or cabinets [149]. If a folder does not exists, it is automatically created when the data is stored into it. A TACOMA agent executing on one host moves to another host by using TCP/IP to communicate with TACOMA software at the destination host. TACOMA agents are migrated using a simple primitive called meet [149]. A TACOMA agent causes another agent to be executed by invoking the meet operation and naming a target agent and a briefcase. The effect of the operation is to terminate the agent invoking the meet and then start executing the target agent with a specified briefcase. Service agents are passively waiting to be activated by a meet. This is somewhat equivalent with the server blocking while waiting for an incoming request in the client-server model. In its simplest form, this delivery is viewed as a procedure call. The folders of the briefcase are equivalent to the arguments of a procedure call, and the agent receiving that briefcase is equivalent to a procedure. The receiving end of remote meet, is the bridgehead, which consists of the firewall that is the entry point to a host. Other entities include guardian processes, a cryptographic service agent, and the individual code service agents.
2.2.5 PMADE

The Architecture of PMADE [2] consists of mainly two components - Agent Host and Agent Submitter. Each host of the network has a server called Agent Host (AH), which accepts and executes incoming agents and a client called Agent Submitter (AS), which submits the agent on behalf of the user to AH.

When a user wants to perform a task, he/she submits the agent, designed to perform that task, to AS on the user system. The AS establishes a connection with the specified AH, where the user is registered, submits the agent and goes offline. The AH examines the nature of the agent and if required, clones and forwards it to other active AHs in the network. It then goes on to execute one clone.

The execution of the agent depends on its nature and state. It can be transferred from one AH to another whenever required. On completion of execution, it submits its results to the AH, which in turn stores the results, until the AS receives them for the user. AS and AH are discussed in detail in the following sections:

The AS plays a crucial role in formulating and dispatching agents to AH. It acts as an interface between the user and AH. One of its primary tasks is to attach a signature to the invoked agent. It retrieves the static IP address of the host on which it is running and binds it to an agent signature.

The AS also receives replies from the AH for user requests. It keeps track of agents and maintains their profile that it submits to the AH from the user’s system. The architecture allows a user to go offline after submitting its agents and receive results on reconnection to the network. The AS functions are as follows:

- Receives an agent and verifies whether it is listed in its Database
• If found, checks the status of AH, connects to it, and sends the agent to
the AH
• If connection is unsuccessful sends an appropriate reply to user
• If agent reaches AH successfully, receives acknowledgment from the AH
• Receives results from AH and stores on the local disk for future
references by the client

The AH is the key component of PMADE. It consists of the manager
modules and the Host Driver. The Host Driver is the basic utility module,
which lies below the manager modules and is responsible for driving the AH,
by ensuring proper coordination between the various managers and making
them work in tandem.

Various manager modules help to perform functions like- agent transfer,
execution, communication, etc. Details of the managers and their functions
are provided in [2]. PMADE provides weak mobility to its agents and allows
one-hop, two-hop and multi-hop agents [2]. PMADE has focused on
Flexibility, Persistence, Security, Collaboration, and Reliability [3].

These agents are grouped into four categories namely- Communication
Managers (Agent Manager, Network Address Manager, Agent Reply
Manager), State Managers( Task Manager, Signature Manager, Cloning
Manager, Resource Account Manager), Persistence Manager( To provide
fault tolerance using checkpointing [152-154]), and Security Manager.

It implements mechanisms to protect local resources from malicious
agents, agents from malicious hosts and for communication among agents
across the network and in local host entities. A large variety of mechanisms,
policies and tools are available in PMADE to achieve flexible levels of
security. These properties make it suitable for the design and implementation
of distributed services in several application areas, viz., mobile computing,
distributed database retrieval [155], network management [156], [157], and information distribution [158]. Details about the security manager of PMADE are given in [159]. Here it suffices to say that a User Manager is used which is responsible for authenticating a user’s account. It deals with the identification and verification of users on the AH. It is invoked when an AH receives a connection request from an AS/AH and sends an acknowledgement to the requesting AS/AH. It keeps records of all user certificates in a database.

2.3 Summary
In this chapter, we have presented the issues of load balancing, fault tolerance, resource discovery, service discovery, security and intrusion detection for different types of networks. We also reviewed some existing MASs.

In the next chapter, we will present Secure and Fault Tolerant Load Balancing Multiagent System architecture.