A DYNAMIC AND SCALABLE PEDESTAL FOR PEER-TO-PEER NETWORKS

This chapter presents a dynamic and scalable pedestal (DSP) for Peer-to-Peer (P2P) network and distributed computing environment. In DSP mainly MAs are used to manage the networks. It provides common solution to P2P system’s fault tolerance and load balancing problems and gives true distributed computing environment with the help of MAs. DSP supports code mobility over the mobile/fixed peer device. We also present a comparative study of the DSP NADSE, Gnutella, and Freenet. Results show that DSP improves the performance of NADSE when number of nodes in network is very large.

Rest of the chapter is organized as follows. Issues are explored in Section 4.1. Section 4.2 highlights on challenges. System model is presented in Section 4.3. Section 4.4 gives the system architecture of DSP. Section 4.5 presents a comparative study of the DSP NADSE with Freenet and Gnutella under the leadership of DSP. Section 4.6 discusses the outcome of the implementation. Finally chapter is summarized in Section 4.7.

4.1 ISSUES

Peer-to-peer (P2P) is a computing model in which peer nodes collaboratively perform a computing task. These peers can serve as both clients and servers and eliminate the need for a centralized node. More simply, a P2P network links the resources of all the nodes on that network and allows the resources to be shared in a manner that eliminates the need for a central host.

The claim for P2P architecture is that enables true distributed computing, creating networks of computing resources. Hosts that have traditionally been used as clients can act as both clients and servers. P2P allows systems to have temporary associations with one other for a while, and then separate. Besides,
nodes in P2P systems are autonomous in the sense that: (i) they can join the system anytime, (ii) they can leave without any prior warning, and (iii) they can take routing decision locally in an ad hoc manner. P2P Unlike the conventional centralized systems, P2P systems offer scalability and fault-tolerance. It is a feasible approach to implement global-scale systems such as the Grid.

Napster adopts central database approach for song titles, but it has inherent reliability and scalability problems that make it vulnerable when there are attacks on the database. Another approach, at the other end of the spectrum, is for the consumer $S$ to broadcast a message to all its neighbors with a request for $F$. When a node receives such a request, it checks its local database. If it has $F$, it responds with the item. Otherwise, it forwards the request to its neighbors, which execute the same protocol. Proceeding in this manner will ensure that a requested resource is always being found when it exists. However, this solution has some critical limitations such as large overhead produced and the looping problem.

Gnutella is a system approach for distributed data management that is based on this idea with some mechanisms to avoid request loops. It uses the time-to-live (TTL) flags in the request message to limit the broadcast scope of the message. However, this scoped broadcast approach does not scale either because of the bandwidth consumed by broadcast messages and the computing cycles consumed by the many nodes that must handle these messages. In fact, the day after Napster was shutdown, reports indicated that the Gnutella network collapsed under its own load, created when a large number of users migrated to Gnutella for sharing MP3 music files. To reduce the cost of broadcast messages, several other studies have been proposed in the literature to support intelligently forwarding and directed bread first search. Many variants of the well-known depth first search (DFS) have been also proposed. Many of the current popular systems, such as KaZaA, which are all based on the FastTrack platform, adopt DFS concept. However, the disadvantage of these approaches, which are considered as hierarchical, is that the nodes higher in the tree take a larger fraction of the load than the leaf nodes, and therefore require more expensive hardware and more careful management. The failure or removal of the tree root or a node sufficiently high in the hierarchy can be catastrophic for the stability of the system.
But the existing P2P systems take more computing time for finalizing the task. For the limitation of processing time of a request we need a system which must support device and computation mobility as per need of the application.

4.2 CHALLENGES
What are challenges in the life of a common person? It is well depicted in Figure 4.1, which shows that whole network is a setup of P2P network. There are several unsolved questions in the mind of a common user of this network. Few are as follows. How interconnected networks will be used to manage the network traffic, Distributed data, Mobile workers, Business extranets, Remote access, Web services, Wireless network, and Mobile smart devices, etc?

![Figure 4.1. Life in a Highly-Connected World](image)

The current available solutions for structured and unstructured P2P have advantages and severe limitations with regard to performance issues. A better P2P performance can be achieved with a more flexible and intuitive architecture which should be well-researched. Thus, we need to design an adequate P2P system which should meet requirements/challenges for fulfilling need of a successful resource management solution.
4.3 SYSTEM MODEL

We are required to develop a computing/communication P2P systems that fulfills most of the above the challenges. The developed system should enable the fast and cost-efficient deployment of self-managed computing/communication devices with high overall management cost, but with low management cost at each peer. With developed system one should be able to deploy large scale computing/communication systems without the need of cost-intensive supercomputing infrastructure in which management is highly complex and requires high-skilled administrators for their maintenance. This system should facilitate to improve the performance and incorporate new ideas.

When NADSE bridges two networks in that situation it should maintain services available in the network along with route information to the node maintaining available services. For this purpose NADSE implements adaptive manager. Architecture of adaptive manager is discussed in the next section.

4.4 DSP ARCHITECTURE

A “Neighbor Assisted Distributed and Scalable Environment (NADSE)” supports based network both device and code mobility. A mobile device will be the member of a cluster. A mobile node in a cluster will work like cluster head (CH) and maintains information about other members of the cluster in the form of database. When a mobile node wants to search some information it requests to CH for members information (viz. IP address, identification certificate, etc.). If the CH is not aware about availability of the type of services a mobile node is interested and presence of the same in the cluster then it guides the same to the mobile node. Then mobile node uses a dynamic and Scalable pedestal (DSP) for load balancing for P2P systems and creates a MA to perform its desired task in the present cluster.

Architecture of a dynamic and scalable pedestal (DSP) system for P2P systems is given in Figure 4.2. DSP provides common solution to P2P system’s fault tolerance and load balancing problems and gives true distributed computing environment with the help of MAs. NADSE (“Neighbor Assisted Distributed and Scalable Environment”) on support of DSP provides fault tolerant and scalable device and code mobility. A mobile device will be the member of a cluster. A mobile node in a cluster will work like cluster head (CH) and maintains
information about other members of the cluster in the form of database. When a mobile node wants to search some information it requests to CH for members information (viz. IP address, identification certificate, etc.). If the CH is not aware about availability of the type of services a mobile node is interested and presence of the same in the cluster then it guides the same to the mobile node. DSP permits a mobile node to create a mobile code for performing its desired task in the present cluster. Further, if the requested task is not completed with members of the present cluster mobile node may move to next cluster or it may take help of DSP running at the CH for multicasting the mobile code to the CHs in the network. After completion of the task final result reaches to the mobile node which was its launching station.

At present DSP contains mainly five DSP agents- DSP Mapping Agent (DMAPA), DSP Route Estimating Agent (DREA), DSP Migration Planning Agent (DMPA), DSP Code Container Agent (DCCA) and DSP Result Container agent (DRCA) in future number of agents may be increased as per need of the applications, means developed system is adaptable in nature. These agents are called DSP agents because at any moment of time as per requirement of the applications algorithm/protocols associated with these agents are changed/updated or new amendment can be made. For balancing the load over the network these agents work together as a DSP multiagent system. DRCA and DCCA facilitates distributed environment for adapting the nature of the network bandwidth. These agents are identified as single entity known as adaptation manager (AM). AM is named because of its nature to accommodate any kind changes occurring in the system. Other component of DSP is Network manager (NM) which is responsible to identify the topology of the network with assistantship of DMAPA. NM provides global identification to mobile devices and MAs. DSP supports code mobility over the mobile/fixed peer device. Architecture of the functioning of the DSP multiagent system is given in Figure 4.2. Brief introductions of these agents are as follows.

4.4.1 DSP Mapping Agent (DMAPA)
It is used by a MA to locate services and to access information on network connection qualities. Connection qualities are especially important for the DREA or and the DMPA to achieve optimizations. In addition to throughput, latency and
other network status information, this agent collects and distributes information on application-level services provided by the CH in the network. DMAPA cares for precise and up-to-date knowledge (maps) within its local CH and provides a rough summarized view of the linked remote CHs. Utilizing the service descriptions in those maps, a MA is able to locate points of interest within the network and see changes in the network structure. Once a list of interesting CH has been determined, another system component - the DREA as shown in Figure 4.2 - can be used by the agent to plan a route [70, 71]. DREA calculates the shortest trip through the net based on the map data.

4.4.2 DSP Route Estimating Agent (DREA)

After getting the list of interesting CH that has been determined by the DMAPA, another system component – the DSP Route Estimating Agent (DREA) as shown in Figure 4.2 - can be used by the agent to plan a route for assigned task. DREA calculates the shortest trip through the net based on the map data. It uses the classic local optimization algorithms. If necessary, a route can be recalculated and amended, for example, in the case of changes in the network or when the agent moves into new CHs and thus shifts its focus with regards to the fisheye paradigm. It facilitates to the MAs for optimizing the sequence of CH to visit, i.e., the route. If an agent chooses a random path through a network, i.e., if an agent is visiting few nodes in cluster then visiting next cluster nodes and in future looking into the previously visited clusters once again then the sequence may lead to a non-optimal total migration time. The route estimating process itself is basically the Traveling Salesman Problem, which is a NP-complete type of problem. But dividing the problems into set of small subsets of problems eases the complex problems. For optimal migration time agent creates a clone for a cluster and number of clones being equal to the number of clusters of interest where the required services may be available. As a consequence, getting an optimal solution in practical application is ruled out. But there are heuristic algorithms (such as local search, genetic, simulated annealing, neural network algorithms, etc.) that have been supporting this working style of agents applied extensively for solving such problems.

The computation of a route is based on the map data. We calculate a kind of DSP Table (DSPT) simply by using the reciprocal values of measured bandwidth.
This matrix has to be updated at regular time intervals to fit the environment’s dynamic behavior. Then, a pathfinder algorithm is applied in order to get a DSPT with shortest paths between two places. In some experiments, we figured out that DSPT is not symmetrically in general. This is caused by variation in the bandwidth values and non-symmetrical connections measured by the DMAPA.

The variation in network throughput influences the result and success of the route estimating, especially short time variations. The DREA generates a route with a fast path through the net on the basis of route matrix. Thereby, some of the best paths may be blocked by short-time traffic. At the point in time, when an agent uses the optimized route, the generated path may not be the best one any more or, in the worst case, is by now the slowest one. The probability that this happens is lower in networks with clearly differing connection qualities. The route estimating is especially useful in networks with different connection qualities and in networks with connections which have different loads over a longer time period. In networks with nearly identical connection qualities, the use of Route estimating algorithms makes no sense – just choose a random path instead of spending time to calculate the random path.

Figure 4.2. Architecture of DSP
4.4.3 DSP Migration Planning Agent (DMPA)

At any point in time, as long as we have a route, a MA may also use a so called Migration Planning Agent (DMPA) as shown in Figure 4.2 to optimize each single migration included in the route. DMPA is mainly designed to reduce network load by selecting and transmitting only those code and data portions of the agent that are needed at the upcoming remote CH. This is, if necessary, done by a concept called slicing or designated code. Other options are to place code in advance in the network, to send data home to carry fewer luggage’s, to change the transmission protocol, etc. An agent may contain one or more task code to be executed at different nodes in the network. The point of time when an agent’s tasks are transmitted depends on the migration strategy, i.e., how a MA is transmitted over the network? There are so called push strategies which transmit an agent’s tasks along with the agent’s state and data before the agent is started at a remote CH. Using a pull strategy, an agent’s tasks are downloaded dynamically while the agent is executed at a remote CH from its home site/DSP Code Container Agent (DCCA). The agent’s home platform is the Agent Submitter where the agent was started first time, i.e., a client equipped with AS. Furthermore, strategies can be distinguished by which tasks are transmitted: all tasks code at once or only some tasks. For example, the pull-all strategy means: transmit an agent, start it at the remote site and in case that at least one task is required; download all tasks of the agent from its home/DCCA. Using a push strategy, agent’s tasks can be transmitted to the next CH of the agent’s route or even to all CH visited by the agent. For example, the push-tasks-to-all strategy transmits first some of agent’s tasks (those tasks which are needed potentially at remote CHs) to all CH which are visited by the agent. Missed tasks will be downloaded dynamically. Then the agent is migrated to the first CH of its route. For the next hops, only the agent is transmitted.

The DMPA is used to optimize time and network load caused by a transmission. This is done by calculating the expected transmission times for different migration strategies. The results are compared to select a best fit migration strategy. It allows us to calculate network load and transmission time for migration of a MA from home network, between CH of its route and back home. For the computation, it takes in account an agent’s size (state, data, and tasks), data which is collected on its route (increases with a constant factor) and
connection qualities (latency and bandwidth). Thereby, a task is used at a remote CH with a certain probability. The data collected by an agent increases by a non constant size and might be transmitted back home from a CH on the agent’s route. There are some technical problems to determine the actual size of an agent at runtime. For the comparison of different migration strategies, this size is constant and needs not to be involved in the computation. The same holds for the collected data. Hence, the DMPA compares the transmission time for the tasks of an agent. The number of tasks and the point in time of transmission differs for different strategies. Possible requests for task downloads have to be taken in account.

In more detail, a computation of the migration time for different migration strategies for a hop is done according to the following scheme: A agent wants to hop from CH \( C_i \) to \( C_{i+1} \). The agent’s home site is \( C_0 \) client’s node. The latency between two CH is defined by the function \( \delta \). Function \( \tau \) denotes the available bandwidth between two CH. The amount of bytes which will be transmitted is \( B_c \) (size of all tasks) for push-all-to-next is \( T = \delta(C_i, C_{i+1}) + B_c / (\tau(C_i, C_{i+1})) \) and for pull-all is \( T = \delta(C_0, C_{i+1}) + B_c / (\tau(C_0, C_{i+1})) \).

Furthermore, it is difficult to determine the probability for the usage of a certain task at a remote CH it is not designated if it is designated it can be very easily traced with the database mapping. Thus, we decided to use the worst case assumption that every task has to be downloaded as long as we do not have any other options. A time computation can be made by pull-tasks

\[ T = \sum_{k=1}^{n} \delta(C_0, C_{i+1}) + \left( B_{c}^{k} + B_{c} \right) / (\tau(C_0, C_{i+1})) \], where \( B_{c}^{k} \) is the size of the \( k \)th task code of the agent. \( B_{c} \) denotes the size of a request for downloading a certain task code.

### 4.4.4 DSP Code Container Agent (DCCA)

It contains all tasks of an agent. The CH is used to serve as DCCA. Such a server can be used by an agent to download tasks instead of downloading from home site. An automatic DCCA initialization might be useful in a case where it takes more time to download tasks from the home site than from a near DCCA with a fast connection. Such a DCCA is the code base for further migrations as long as there is a good connectivity. This is useful only for pull strategies (downloading tasks code dynamically). The optimization is simply based on a comparison of
migration times with and without a DCCA initialization. With a low optimization degree, the module compares the migration time with the home site as a DCCA and with a local DCCA on the current CH. A medium optimization degree is reached, if all available DCCAs are taken into account. As a variation of the low degree optimization, the migration times for further migrations with a dynamic DCCA initialization are computed (high optimization degree).

### 4.4.5 DSP Result Container Agent (DRCA)

It is used by an agent to upload collected data instead transmitting data to the home site. The initialization of a DRCA depends on whether an agent wants to transmit collected data to home site. Collected data loads the network again and again when the agent migrates. Where a MA does not need this collected data for further computations, the data should be sent home site. DMPA computes that whether it is cheaper to initialize a DRCA to upload data instead of using home site to upload data. Automatic data upload variant calculates the migration time to the next CH, if all data is carried along with the agent. The result is compared with the time to upload collected data and to migrate without unnecessary data. An agent can initialize code and DRCAs on its route. With this extended network model, the effort and the advantage of initializing and using code and DRCAs can be computed. The introduced optimization variants are some approaches to reduce network load and migration time of a MA. CH is also used as DRCA.

### 4.4.6 Algorithm

The basic process for an optimization is simple- calculate migration times of different migration strategies. Compare results and choose best migration strategy. A MA might use the DMPA to compute an optimal migration strategy regarding migration times for parts of its route or even for the whole route. A simple variant is to optimize the next hop only by comparing the migration strategies push-all-to-next, pull-all, pull-tasks and push-all-to-all. The algorithm looks like this:

```c
/*Calculate transmission times*/
/*Push-all-to-next: Transmit tasks to next CH*/
```
\[ T-patn = \text{delay}(c_i, c_j) + \frac{\text{Task}_\text{size}/\text{bandwidth}(c_i, c_j)}{j_i} \]

/* Pull-all: Download tasks from home site at next CH*/
\[ T-pa = \text{delay}(c_0, c_j) + \frac{\text{Task}_\text{size}/\text{bandwidth}(c_0, c_j)}{j} \]

{Pull-task: Download each task from home at next CH}
\[ T-pt = \text{delay}(c_0, c_j) + \frac{\text{SUM(Probability}(k)\times(\text{Task}(k)+\text{Request}))}{\text{bandwidth}(c_0, c_j)} \]

/* Only for the first hop: push-all-to-all: and Distribute tasks from home to all CH*/

for \( s \) in servers
{
\[ T-pata = T-pata + \text{delay}(c_0, c_j) + \frac{\text{Task}_\text{size}/\text{bandwidth}(c_0, c_s)}{j} \]
}

/* Select migration strategy */
\[ T-min = T-patn; \]
\[ MS = "push-all-to-next"; \]
if ( \( T-pa < T-min \) )
{
\[ T-min = T-pa; \]
\[ MS = "pull-all"; \]
}
else if ( \( T-pt < T-min \) )
{
\[ T-min = T-pu; \]
\[ MS = "pull-tasks"; \]
}
else ( \( T-pata < T-min \) )
{
\[ T-min = T-pata; \]
\[ MS = "push-all-to-all"; \]
}

The migration strategy push-all-to-all can be used only at the home site. From there, all tasks are transmitted to all CH visited by the agent. Then, only the agent
needs to be transmitted between the CH of the route. No additional tasks are necessary. A special case is also the last hop of a MA. This is the migration back to the home site. Thereby, the collected data and the agent are transmitted only. Thus, there is no optimization for this hop.

A similar optimization variant is to optimize the migration for more than one hop (not only for the next hop). The computation of transmission times is made for all migrations. Thereby, the migration strategy is fixed for all hops. This method can be improved, if the migration strategy is not fixed at all. The complexity of the computation is increased for this method. We have to check this method in more detail before we implement it. Two optimization variants include by the DMPA are: code and mirror server.

4.4.7 Analysis of Algorithm
The DMAPA is used by a MA to locate services and to access information on network connection qualities. Connection qualities are especially important for the DREA and the DMPA to achieve optimizations. DMAPA cares for precise and up-to-date knowledge (maps) within its local domain and provides a rough, summarized view of the linked remote CHs. It consists of several network servers for computation. A MA is able to locate points of interest within the network of CH. Once a list of interesting CH has been determined, another system component - the DREA can be used by the agent to plan a route. As long as we have a route, a MA may also use a so called DMPA.

For computing the overhead on the network we have taken few assumptions which are: Total Nodes = \( T_N \), Nodes of Interest = \( I_N \), Packets per Node = \( N_P \), Size of Packet = \( S_p \), Requested Packet = \( R_p \), \( u = \) Requested Packet (\( R_p \)) / Total Packets Available ((\( N_p \))), and Traffic due to one node = \( N_pS_p \). Thus, Total Traffic due to all the nodes of Interest can be given by \( T_{i,N} = \sum_{i=1}^{I} N_pS_p \) and Traffic due to relevant Packet on network will be \( T_{i,r} = uS_p \). Overhead (\( O_{MM} \)) can be defined as (Total Traffic due to all nodes of Interest - Traffic due to Relevant Packet on network) is given by \( \sum_{i=1}^{I} N_pS_p - uS_p \Rightarrow S_p(\sum_{i=1}^{I} N_p - u) \).
DREA component is able to calculate the shortest trip through the net based on the map data. This module may be used by MAs to optimize the sequence of CH to visit, i.e., the route. Route is based on the map data. This component uses classic local optimization algorithms. Minimum path can be found out with the help of route matrix. DSPT is calculated simply by using the reciprocal values of measured bandwidth and this matrix has to be updated dynamically. A path finder algorithm is applied in order to get a DSPT with shortest paths between two places. We have assumed $\tau_i \rightarrow$ be measured bandwidth of $i^{th}$ node and where $i = 1$ to $N$, the DSPT $D_i$ for $N$ is given by

$$
D_i = \begin{bmatrix}
a_{i1} & a_{i2} & \cdots & a_{iN} \\
a_{2i} & a_{22} & \cdots & a_{2N} \\
\vdots & \vdots & \ddots & \vdots \\
a_{Ni} & a_{N2} & \cdots & a_{NN} 
\end{bmatrix}
$$

Thus, $a_{ii} = 1/\tau_{ii}$ where $a_{ii}$ is calculated from the reciprocal value of $\tau_{ii}$. Where $a_{ij}$’s stands for places or point of interest for which we find out the shortest route. Assume $P_i$ be a DSPT that is find out from $D_i$ by applying a path finder algorithm. Elements of $P_i$ gives us the shortest path between two places and $b_{ij}$’s are the elements of matrix $P_i$. $b_{ii}$’s stands for shortest path between two places.

$$
P_i = \begin{bmatrix}
b_{i1} & b_{i2} & \cdots & b_{iN} \\
b_{2i} & b_{22} & \cdots & b_{2N} \\
\vdots & \vdots & \ddots & \vdots \\
b_{Ni} & b_{N2} & \cdots & b_{NN} 
\end{bmatrix}
$$

DSPT is not symmetrical because of varying bandwidth values and non symmetrical connections measured by the DMAPA. For TSP, there are algorithms for asymmetrical and for symmetrical matrices. As all of the working of DREA is done locally so no traffic is present on the network to go from one place to another only algorithms are used for local calculations of path so we assume the overhead is negligible in this case.

During execution, an agent consists of three parts: the agent’s state and data and its set of tasks. As long as we have a route, a MA may also use a DMPA to
optimize each single migration included in the route. DMPA is mainly designed to reduce network load by selecting and transmitting only those code and data portions of the agent that are needed at the upcoming remote CH. This is also used to optimize time during transmission. Network load is the traffic on the network due the migration of MAs. Total traffic and overhead is calculated as given below so that we optimize the network load. MA carries its code and state across the network. For each hop $j$ traffic is $T_{MP}^j$.

Traffic generated due DMPA is given by $T_{MP}^j = R_p C_{MA} S_j^j S_p$, where $R_p \to$ Requested Packet, $C_{MA} \to$ Code for MA, $S_j^j \to$ Size of the state of agent at hop $j$, and $S_p \to$ Size of a Packet. Thus, size of state of the agent is given by $S_j = d_{list} + \omega + \sum_{i}^j u S_p$, where $d_{list} \to$ Size of the list, $\omega \to$ Size of the other internal data structure representing the state of computation $\sum_{i}^j u S_p \to$ Indicate the useful information collected by the agent at each, Visited node. $d_{list}$, $u$, $S_p$ and $\omega$ do not depend on the node and for simplicity $\overline{\omega} = d_{list} + \omega$. Thus Overall Traffic $T_{MP} = \sum_{j=0}^{N} (R_p S_p + C_{MA} + \overline{\omega} + \sum_{i}^j u S_p)$ and Traffic Overhead are given by $O_{MP} = T_{MP} - T_{IP}$, i.e.,

$\left( R_p S_p + C_{MA} + \overline{\omega} \right) (I_N + 1) + (1/2)(I_N + 1) - I =>$

$\left( R_p S_p + C_{MA} + \overline{\omega} \right) (I_N + 1) + (1/2)(I_N - 1)$

### 4.5 Implementation and Performance Study

For testing NADSE we have used total 25 nodes (2.2 Core 2 Due processor, 1 GB RAM, 160 GB HDD, Windows-XP, Java SDK 1.5), one server (for Internet communication), 1 access point (2700 DLink), 6 routers (2 No. CISCO 2851, 2 No. CISCO 2811, and 2 No. CISCO 1841) dividing 24 nodes into six networks of categories (Class A, Class B and Class C). 25th node used to take the services of the global network.

Here in this setup this node (25th) used to move the MA on the infrastructured network, i.e., wired network. Complete setup is wireless. Further it is also considered that same node may be considered for multiple times for increasing the number of nodes in the system or MA may visit infrastructure network depending
on the availability of the services. This node also maintains list of available services and route to destination where services are available.

We have tested DSP on the NADSE for searching and downloading an information file over/from the network when NADSE services are active and not active. Figure 4.3 presents the time consumed in the completion of task (distributing an information file of size 512 KB). From the result it is clear that Freenet and Gnutella takes almost same time but DSP NADSE is superior in term of overall performance. Reason behind the better performance is MA which distributes the task in parallel by cloning itself and collects the outcome of the clones.

Figure 4.3 Distributing an Information file which is distributed over several nodes when NADSE service node is active.

Figure 4.4. Distributing an Information file which is distributed over several nodes when NADSE service node is not active
Figure 4.4 presents the time consumed in the completion of task (distributing an information file of size 512 KB) when NADSE service is not active. From the result it is clear that Freenet and Gnutella takes almost same time but due to unavailability of NADSE service maintenance node NADSE network takes more time in comparison to NADSE service provider. It is found that still NADSE based network is superior in term of overall performance. Reason behind the better performance is MA which distributes the task in parallel by cloning itself and collects the outcome of the clones.

Figure 4.5 presents the time consumed in the completion of task (searching/downloading an information file of size 512 KB). From the result it is clear that Freenet and Gnutella takes almost same time but NADSE is superior in term of overall performance. Reason behind the better performance is MA which distributes the task in parallel by cloning itself and collects the outcome of the clones.

Figure 4.5. Searching an Information file which is distributed over several nodes when NADSE service node is active

Figure 4.6 presents the time consumed in the completion of task (searching/downloading an information file of size 512 KB) when NADSE service is not active. From the result it is clear that Freenet and Gnutella takes almost same time but due to unavailability of NADSE service maintenance node NADSE
network takes more time in comparison to NADSE service provider. It is found that still NADSE based network is superior in term of overall performance. Reason behind the better performance is MA which distributes the task in parallel by cloning itself and collects the outcome of the clones. Further NADSE uses direct communication channel to send the data/information to the requesting node.

Figure 4.6. Searching an Information file which is distributed over several nodes when NADSE service node is not active

4.6 RESULT AND DISCUSSION
NADSE uses DSP for distributing the task in the network. Further NADSE is based on the cluster and network is divided into small subnetworks. This division of the network does not increase more traffic when parallel computation of a task is done. Only traffic increased in a cluster which is being very small network and not affected by the network traffic. When NADSE bridges two networks in that situation it maintains services available in the network along with route information to the node maintaining available services. Comparative results show that NADSE network performs better in comparison to Gnutella, and Freenet. From results it is also clear that DSP improves the performance of NADSE when number of nodes in network is very large.

4.7 Summary
In this chapter we have presented a Dynamic and Scalable Pedestal (DSP) for P2P systems. In this system mainly MAs are used to manage the network. DSP
provides common solution to P2P system’s fault tolerance and load balancing problems and gives true distributed computing environment with the help of MAs.

In the next chapter MA (MA) based resource management in Distributed Computing Environment (DCE) is presented.