CHAPTER-5
MANAGING VOLATILE WEB CONTENTS USING MIGRATING AGENTS

5.1 INTRODUCTION

Today the web has grown exponentially in size, and as of today it contains billions of publicly indexable web documents distributed all over the world on thousands of web servers. Large size and the absence of centralized control over its contents are two key factors for the success of the web. In-fact quality is distributed very off-centered i.e. interesting pages are sparse in comparison with the rest of the contents. Hence, there is a need of a more effective way of retrieving information from the web. To cope up with this sheer volume of data available on the web, programs that run off of special websites called search engines; has been designed. Search engines help the user to find relevant pages from the web based on important words. A crawler downloads web pages from the web to be used by search engine later.

The centralized crawling techniques are unable to cope up with constantly growing web. Distributed crawling methods based on migrating (or mobile) agents is an essential tool for allowing such access that minimizes network utilization and also keeps up with document changes. Using migrants (migrating crawlers), the process of selection and filtration of web documents can be done at web servers rather than search engine side which can reduce network load caused by the web crawlers.

5.2 MIGRATING CRAWLING AGENTS

Agent is an autonomous entity that acts on behalf of others in an autonomous fashion, performs its actions in some level of proactivity and reactivity, and exhibits some levels of the key attributes of learning, co-operation and mobility. Agents can be classified according to the actions they perform, their control architecture, the range and effectiveness of their actions, the range of sensitivity of their senses, or how much internal state they posses. Migrating agents are computational software processes capable of roaming wide area networks such as the web, interacting with foreign hosts, gathering
information on behalf of its owner and reporting back after performing the duties assigned by its master.

Mobility allows an agent to move or hop, among agent platforms. The agent platform provides the computational environment in which an agent operates. The platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. One or more hosts may comprise an agent platform, and an agent platform may support multiple computational environments, or meeting places, where agents can interact. They may cooperate or communicate with other agents making the location of some of its internal objects and methods known to other agents without necessarily giving all its information away.

The key characteristics of mobile agents are: migration, data acquisition, route determination, and communication. Some of the advantages of mobile agents are bandwidth, latency, asynchronous task execution, fault tolerance, and peer-to-peer communication. The trends outlined in the previous section lead to the conclusion that mobile code, and mobile agents, will be a critical near-term part of the Internet. Not because mobile code makes new applications possible, nor because it leads to dramatically better performance than (combinations of) traditional techniques, but rather because it provides a single, general framework in which distributed, information-oriented applications can be implemented efficiently and easily, with the programming burden spread evenly across information, middleware, and client providers.

Distributed systems often rely on communication protocols that involve multiple interactions to accomplish a given task. This results in a lot of network traffic. Mobile agents allow packaging a conversation and dispatching it to a destination host where the interactions can take place locally. Mobile agents are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather than to transfer over the network. The motto is to move the computations to the data rather than the data to the computations.

The agent approach use the bandwidth of the network to migrate an agent to a platform, and allows it to continue to run after leaving a node, even if they lose connection with the node where they were created thereby provide the better utilisation of communication and
allows parallel distributed applications. In simple words, an agent can move on to other machines when necessary and can delegate tasks to other mobile agents in order to achieve real parallel applications.

5.3 DISTRIBUTED CRAWLING USING MIGRATING AGENTS

In traditional crawling (see Figure 5.1) the pages from all over the web are brought to the search engine site for processing. When a page is brought to the search engine site and analysed, many a times it is found that it was not needed. In such cases the efforts made to send request to the web server and bringing the page to the search engine site seems to be useless and results a lot of network traffic.

![Figure 5.1: Traditional Centralized Web Crawling](image1)

In the distributed crawling with migrating agents approach (see Figure 5.2), agents allow packaging a conversation and dispatching it to a destination host where the interactions can take place locally.

![Figure 5.2: Onsite crawling using migrating crawling agents](image2)
Migrating agents are also useful when it comes to reduce the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather then transfer over the network. The main concern is to move the computations to the data rather than the data to the computations. By migrating to the location of the resource, a mobile agent can interact with the resource much faster than that across the network which reduces network traffic also as shown in Figure 5.3.

![Diagram showing crawling with migrating agents](image)

*Figure 5.3: Crawling with Migrating Agents*

The distributed crawling with migrating agents (see Figure 5.4) approach uses a crawler manager at the search engine site, that deputes migrating crawlers to the web servers with a list of URLs of respective web servers. The migrating crawler, on reaching a server crawl the pages, selects the best of the pages for its collection and comes back to the search engine with the collection. It reduces unnecessary overhead of being the unnecessary pages to the search engine site. The size of the collection can further be reduced by filtering the required specialized web pages and even compressing them.
Algorithm for distributed crawling with migrating crawler (mc) approach is shown in Figure 5.5.

```plaintext
While (true)
  While (CrUrls1 is not empty)
    url=SelectURLtoCrawl(CrUrls1)
    if  website(url) allows mc
      newpage=page(url)
      newUrls=SelectEmbedUrls(newpage)
      newUrlsQ=newUrlsQ U newUrls
      process(newpage)
      collofpages= collofpages U newpage
    while newURLsQ is not empty
      url=SelectURLtoCrawl(newUrlsQ)
      newpage=page(url)
      newUrls=SelectEmbedUrls(newpage)
      newUrlsQ=newUrlsQ U newUrls
      process(newpage)
      collofpages= collofpages U newpage
      compress(collofpages)
      AllUrls = AllUrls U newUrlsQ
    CrUrls1= (CrUrls1 – url ) U newUrlsQ
    CrUrls2= CrUrls2 U url U newUrlsQ
```

**Figure 5.5:** Algorithm for Distributed Crawling using Migrating Agents
Where,

AllUrls=List of all Urls

CrUrls1=List of Urls to be crawled

CrUrls2=List of Urls already crawled

On analysing the above approach using a set of data as follows:-.

Let,

The number of HTML pages on a remote site=100

Average size of each HTML page=10KB

The network traffic with a traditional crawler =100x10 =1000KB

The network traffic for a migrating crawler

=100 x 10 =1000KB + 2KB (size of migrating crawler) =1002KB

If 60% of the pages are of use, then network traffic using migrating crawler

=60x10 =600KB+2KB=602 KB

On compressing the collection using WinRR compression technique, it is observed that HTML documents are compressed approximately 80%.

So, now after compression, network traffic using migrating crawler

= 600 x.20=120KB+2KB=122 KB

The analysis shows that the network traffic can be reduced substantially using a migrating crawler. Generally all the pages on a website are of no use and it is unnecessary to bring all pages to the search engine site. If 60% of the pages found on a web server are of use, then the network traffic for a traditional crawler is 1000KB whereas with a migrating crawler it can be reduced to 602 KB, a reduction of almost 60%. Using any compression technique this size can further be reduced to even 80% more that comes out to be a
network load of 122 KB. That decreases the network traffic by almost 88%. Performance analysis with multiple servers (100 pages of 10 KB each, per server) is shown in Table 5.1.

Table 5.1: Performance analysis with Multiple Servers

<table>
<thead>
<tr>
<th>No. of Web Servers</th>
<th>Load with Traditional Crawler</th>
<th>Load with Migrating Crawler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without Compression</td>
<td>With Compression</td>
</tr>
<tr>
<td>01</td>
<td>1000 KB</td>
<td>1002 KB, 202 KB</td>
</tr>
<tr>
<td></td>
<td>800 KB</td>
<td>802 KB, 162 KB</td>
</tr>
<tr>
<td></td>
<td>500 KB</td>
<td>502 KB, 102 KB</td>
</tr>
<tr>
<td>10</td>
<td>10000 KB</td>
<td>10020 KB, 2020 KB</td>
</tr>
<tr>
<td></td>
<td>8000 KB</td>
<td>8020 KB, 1620 KB</td>
</tr>
<tr>
<td></td>
<td>5000 KB</td>
<td>5020 KB, 1020 KB</td>
</tr>
<tr>
<td>100</td>
<td>100000 KB</td>
<td>100200 KB, 20200 KB</td>
</tr>
<tr>
<td></td>
<td>80000 KB</td>
<td>80200 KB, 16200 KB</td>
</tr>
<tr>
<td></td>
<td>50000 KB</td>
<td>50200 KB, 10200 KB</td>
</tr>
<tr>
<td>1000</td>
<td>1000000 KB</td>
<td>1002000 KB, 202000 KB</td>
</tr>
<tr>
<td></td>
<td>800000 KB</td>
<td>802000 KB, 162000 KB</td>
</tr>
<tr>
<td></td>
<td>500000 KB</td>
<td>502000 KB, 102000 KB</td>
</tr>
</tbody>
</table>
Performance analysis with single server having ten thousand pages is graphically shown in Figure 5.6.

![Network Load Reduction Using Migrating Crawling Agents](image)

**Figure 5.6:** Performance analysis with single website of 10,000 pages

Network load reduction using migrating crawling agents is shown in Figure 5.7.

![Network Load Reduction Using Migrating Crawling Agents](image)

**Figure 5.7:** Network load Reduction using Migrating Crawling Agents

### 5.4 MANAGING VOLATILE INFORMATION

As per Sharma et al approach [33] to manage volatile web data, in a HTML document structure, the `<SPAN>` Tag is a container of any text element offering a generic
mechanism for adding structure to documents. The <SPAN> Tag is primarily intended for specifying layout that can be used to specify volatile information with the help of class attributes: Vol1, Vol2, and Vol5 as follows:-

<BODY>

SHOBHIT UNIVERSITY, MEERUT

ADMISSION NOTIFICATION No. <SPAN Class =Vol1”> M-02/ADM/2012-201 </SPAN>

The complete HTML code containing volatile information using SPAN Tag is shown in Figure 5.8.

```html
<BODY>

SHOBHIT UNIVERSITY, MEERUT

ADMISSION NOTIFICATION No. <SPAN Class =Vol1”> M-02/ADM/2012-201 </SPAN>

Applications in the prescribed forms are invited for the following courses offered at various Faculties of the University for the academic year <SPAN class = “Vol2”> 2012-2013 </SPAN>

Schedule for the Entrance Test:

Date: <SPAN class =“Vol3”> 12.03.2012 and Time : <SPAN class =“Vol4”> 9 A.M. to 5 P.M. </SPAN>,

Venue <SPAN class =“Vol5”> Shobhit University, Meerut </SPAN>

</SPAN>

Figure 5.8: An HTML Document containing Volatile information
Similarly, the XML scheme of document structure offers more flexibility by allowing web page designers to use their own set of markup tags. These tags can be used to reflect the volatile information contained in a document as shown in Figure 5.9.

![Figure 5.9: An XML Document containing Volatile information](image)

It was suggested that the Tags and the information be stored separately in a file having same name but with different extension (say .TVI) as shown in Table 5.2.
Table 5.2: TVI file containing only the Volatile information

<table>
<thead>
<tr>
<th>Vol1</th>
<th>M-02/ADM/2012-2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vol2</td>
<td>2012-2013</td>
</tr>
<tr>
<td>Vol3</td>
<td>12.03.2012 and Time:</td>
</tr>
<tr>
<td>Vol4</td>
<td>9 A.M. to 5 P.M.</td>
</tr>
<tr>
<td>Vol5</td>
<td>Shobhit University, Meerut</td>
</tr>
</tbody>
</table>

This TVI (Table of variable information) file shall be updated every time the changes are made to the hypertext document. Every time when the latest version of the page is required all the Vol# Tags (volatile information tags) can be extracted out from the document along with their associated volatile information for updated information. This file containing the changed contents of a document would definitely be substantially smaller in size as compared to the whole document. For eg, the TVI file for the Figure 5.8 or Figure 5.9 will only contain the information shown in Table 5.2. It has been found that the size of a .TVI file is on an average 5% of the size of its corresponding main hypertext document.

Use of migrants to extract volatile information from the web servers using table of volatile information enables web crawlers to select only modified information and bring it to the search engine side, leaving all other static information which already is in the search engine repository.

In traditional crawling the pages from all over the web are brought to the search engine side and then processed, and only after analyzing the page it can be concluded that whether the page is useful or not. Studies suggest that most of the times, the downloaded page is not useful in the sense that it has not updated since its last crawl. In such cases the efforts made to send an HTTP request to the web server and bringing the page to the search engine side seems to be useless and also causing unnecessary load on to the internet.

In the migrating approach of crawling, migrants are allowed to migrate to a destination host where the interactions, downloading and processing of documents can take place locally on to the web server itself (see Figure 5.10). The main concern is to move the
computations to the data rather than the data to the computations. By migrating to the location of the resource, a migrant can interact with the resource much faster than using HTTP across the network.

Unlike in traditional centralized approach in which all the job is to be done at the search engine side, this approach performs the job of selecting the volatile information at the place where information lies. This process may be implemented in parallel and multiple migrants may be sent to many web servers to maintain the scalability.

In the proposed system (see Figure 5.11) the major components are Crawl Manager and Migrants. The Crawl manager creates multiple worker threads named as Migrants, and supplies to Migrant a set of URLs (SubUrlList). Where each URL in the SubUrlList provides the path to the document. The Migrant is a migrating crawling agent that along with SubUrlList goes to the remote site. It downloads .TVI file containing volatile information and stores all .TVI files to TVICOLL file. This process repeats until SubUrlList is exhausted. It also extracts the internal and external links and adds them to SubUrlList and OuterLinks file.

Figure 5.10: Migration of Migrants to the Web servers
Figure 5.11: Transfer of Migrants and their functioning at the remote site

The algorithm of Crawl Manager is shown in Figure 5.12.
Crawl manager()
{

// manages a list of URLs MainUrlList

SubUrlList=Sublist of Urls selected from MainUrlList

Creates and migrates a Migrant with SubUrlList to Web server

//……………..

// The Migrant goes to the web server

// and extracts volatile information, local links and outer links

//……………..

// On arrival back to the Crawl manager

Changes are updated to the local collection

MainUrlList=MainUrlList U SubUrlList U OuterLinks

}

Figure 5.12: Algorithm for Crawl Manager

The working of a migrant is shown in Figure 5.13.
After filtration and compression, TVICOLL and OuterLinks are sent back to the search engine side. Here the URLs from the OuterLinks are added to the MainUrlList and local repository of the search engine is updated with the TVICOLL as shown in Figure 5.14.
Due to the lack of efficient refresh techniques, current crawlers add unnecessary traffic to the already overloaded Internet. Frequency of visits to sites can be optimized by calculating refresh time dynamically. It helps in improving the effectiveness of the crawling system by efficiently managing the revisiting frequency of a website and appropriate chance to each type of website to be crawled at appropriate rate.

5.5 OBSERVATIONS

To analyze and compare the approaches, a set of three websites are taken into consideration that contained information of volatile nature as well as static nature. First website contains approximately 8% of volatile contents, second containing 5% and third with least volatile data i.e. 2% only. The average size of a HTML page was 180KB so the network traffic caused using traditional centralized crawling approach was 540 KB. Whereas on using migrants, the pages were compressed at the server side and then the traffic load found was 108 KB. On using the proposed approach that collects only the volatile contents from the web pages it was found that the load came down to just 27 KB.

To measure the load in better manner the volatile contents of the pages were modified before every visit that gave slight variation in the size of pages and measurements of the load incurred dye to revisits of the crawler have been taken for each of the three approaches i.e. centralized, migrating and using decentralized approach with migrants & TVI. Load Caused using Centralized approach is shown in Figure 5.15.
Load caused using migrating web crawler is shown in Figure 5.16.

Load caused using with migrants and TVI is shown in Figure 5.17.
It can be observed that, after five visits to the pages the load incurred has been found 2727 KB, 545.4 KB and 136.8 KB respectively and after ten visits the load was 5435 KB, 1092.2 KB and 272.98 KB respectively as shown in the Figure 5.18.
A migrating crawling approach has been used wherein migrants after moving to the web servers downloads the .TVI (table of variable information) file only for maintaining the freshness of search engine repository. The preliminary experimental results show the reduction in network load by a fraction of 95% approximately in comparison to centralized approach and the network load reduced by a fraction of 75% approximately in comparison to migrating crawling approach. The freshness and quality of the collection also get improved significantly at search engine side.

5.6 INDEX CONSTRUCTION

A general web search engine relies on the indexes and on the repository. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, normally incorporated as an inverted index, the search engine would scan every possible document on the web, which would require considerable time and computing power; impossible with the current web size. Such an index determines which documents match a query but does not rank matched documents. In some designs index includes additional information such as the frequency of each word in each document or the positions of a word in each document. Position information enables the search algorithm to identify word proximity to support searching for phrases; frequency can be used to help in ranking the relevance of documents to the query.

Inverted indices are one of the most commonly used techniques to organise very large document collections and provide high-speed access to sets of documents satisfying queries. This index can only determine whether a word exists within a particular document. Search engines return results for a given keyword based on keyword matching and are unable to search for the keywords based on the sense behind the keyword. In this chapter, the need of a perspective term based index construction is proposed, that stores keywords with their synonyms, acronyms and alternate spellings, and searches the related terms based on user perception i.e. the word for which he is intended to search. The perspective term based index will help the search engine designers to provide better results to the user based on his mental vision.

Keyword searching is the most common form of text search on the web. Most search engines do their text query and retrieval using keywords. It can simply be any word on a web page. For example, on using words like "and, is, am or are", making it one of the
keywords for this particular webpage in some search engine's index. However, since the words like "and, is, am or are" have nothing to do with the subject of this webpage, these are not very useful keywords. Useful keywords and key phrases for a page would be “Sachin Tendulkar”, "Bill Gates", "Prime Minister", "medicine", "cricket", "Maruti", "Hyundai", or “Bajaj auto” etc. These keywords would actually tell a user something about the subject and content of a page.

While searching with most popular search engines Google, Yahoo and Bing it is found that these search engines provide millions of related URLs for similar searches. On searching for Microsoft’s chairman Mr. Bill Gates, depending upon the way in which query is fired, Google returns .663 million to 106 million results, Yahoo returns 4.93 million to 301 million results, and Bing returns 2.47 million to 75.8 million results as shown in Table 5.3.

<table>
<thead>
<tr>
<th>Query fetched</th>
<th>Google</th>
<th>Yahoo</th>
<th>Bing</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill gates</td>
<td>106,000,000</td>
<td>301,000,000</td>
<td>75,800,000</td>
</tr>
<tr>
<td>chairman Microsoft</td>
<td>33,400,000</td>
<td>13,900,000</td>
<td>13,500,000</td>
</tr>
<tr>
<td>president chairman Microsoft</td>
<td>31,000,000</td>
<td>9,400,000</td>
<td>9,230,000</td>
</tr>
<tr>
<td>Microsoft president chairman</td>
<td>19,100,000</td>
<td>9,390,000</td>
<td>9,240,000</td>
</tr>
<tr>
<td>chairman president Microsoft</td>
<td>19,100,000</td>
<td>51,000,000</td>
<td>9,140,000</td>
</tr>
<tr>
<td>Microsoft chairman</td>
<td>10,400,000</td>
<td>17,100,000</td>
<td>13,600,000</td>
</tr>
<tr>
<td>bill gates chairman Microsoft</td>
<td>6,210,000</td>
<td>15,040,000</td>
<td>2,470,000</td>
</tr>
<tr>
<td>bill gates chairman president Microsoft</td>
<td>3,830,000</td>
<td>63,500,000</td>
<td>2,890,000</td>
</tr>
<tr>
<td>Microsoft chairman president bill gates</td>
<td>789,000</td>
<td>50,600,000</td>
<td>2,850,000</td>
</tr>
<tr>
<td>Microsoft chairman bill gates</td>
<td>663,000</td>
<td>14,930,000</td>
<td>2,530,000</td>
</tr>
</tbody>
</table>

Table 5.3: Search results Microsoft’s chairman Mr. Bill Gates
It shows while user wishes to search for Bill Gates but passes the query in different forms, like Bill gates, Microsoft Chairman or Microsoft Bill Gates, the search engines matches the keywords extracted from the documents and shows the list of documents that contain the given string as shown in Figure 5.19.

![Diagram](image)

**Figure 5.19:** Keyword matching for Microsoft’s chairman Mr. Bill Gates

Having surfed the Google for various types of people, it is also found that Google shows suggestions for next keyword. On typing a word it shows a list of ten next relevant words from which user may select a word, and same remains in continuation till the query is completed. For example on typing “who”, Google displays a suggestive list of keywords as shown in Figure 5.20.

![Google Suggest](image)

**Figure 5.20:** Suggestive keywords for keyword “who”
Whereas, on further typing “believes” it displays a list of more suggestive keywords as shown in Figure 5.21.

![Google Suggest](image)

**Figure 5.21:** Suggestive keywords for keyword “who believes”

While searching for Atheist, on passing string “Who believes in God”, Google returns 186,000,000 results, out of which majority of results are irrelevant as shown in Figure 5.22.

![Google Search Results](image)

**Figure 5.22:** Results returned for keyword “who believes in God”

Whereas, on passing string “Theism” for which a user would be intended to search, Google returns only 2,870,000 results as shown in Figure 5.23.
Similarly, while searching for Cannibalism, on typing “who eats”, Google displays a suggestive list of keywords as shown in Figure 5.24.

Further, on passing string “Who eats human flesh“ Google returns 6,100,000 results, out of which majority of results are irrelevant as shown in Figure 5.25.
Figure 5.25: Results returned for keyword “who eats human flesh”

Whereas, on passing string “Cannibalism” for which a user would be intended to search, Google returns only 9,240,000 results as shown in Figure 5.26.

Figure 5.26: Results returned for keyword “Cannibalism”

Further, on passing string “who knows everything” Google returns 486,000,000 results, out of which majority of results are irrelevant as shown in Figure 5.27.
Figure 5.27: Results returned for keyword “who knows everything”

Whereas, on passing string “Omniscience” for which a use would be intended to search, Google returns only 4,980,000 results as shown in Figure 5.28.

Figure 5.28: Results returned for keyword “Omniscience”

Further, on passing string “one who loves himself”, Google returns 141,000,000 results, out of which majority of results are irrelevant as shown in Figure 5.29.
Whereas, on passing string “Narcissism” for which a user would be intended to search, Google returns only 8,240,000 results as shown in Figure 5.30.

From the above analysis it is found that on passing similar queries, Google returns a large number of results of which most of them are of no use (see Table 5.4).
Table 5.4: Number of results returned by Google for various similar queries

<table>
<thead>
<tr>
<th>Query fetched</th>
<th>Google</th>
</tr>
</thead>
<tbody>
<tr>
<td>who believes in god</td>
<td>186,000,000</td>
</tr>
<tr>
<td>Theism</td>
<td>2,870,000</td>
</tr>
<tr>
<td>who eats human flesh</td>
<td>6,100,000</td>
</tr>
<tr>
<td>Cannibalism</td>
<td>9,240,000</td>
</tr>
<tr>
<td>who knows everything</td>
<td>486,000,000</td>
</tr>
<tr>
<td>Omniscience</td>
<td>4,980,000</td>
</tr>
<tr>
<td>one who loves himself</td>
<td>141,000,000</td>
</tr>
<tr>
<td>Narcissism</td>
<td>8,240,000</td>
</tr>
</tbody>
</table>

From the above results it is evident that any general search engine does not generate results as per perception of the user, rather these perform just keyword matching and results are generated on ranking basis. It may be assumed that keywords are arranged in a tree like structure as shown in Figure 5.31.

![Keyword matching for “who eats human flesh”](image-url)

**Figure 5.31**: Keyword matching for “who eats human flesh”
Here interrelated words are arranged in a tree structure, and when a user enters a keyword, it searches the tree for that keyword and shows interrelated keywords to the user, and proceeds further. For example, in Google, on typing keyword “who eats human”, Google automatically displays flesh.

### 5.7 PERSPECTIVE TERM BASED INDEX

The proposed approach needs a mechanism through which a search engine stores keywords with their synonyms, acronyms and alternate spellings. They should also store interrelated words in which user may be interested in and store them in priority of order in which user normally search them. It will allow the user to search the index not only as per the keywords but also as per the view of the user. For example keyword1(KW1) may further be followed by KW2, KW3 and KW4 (see Figure 5.32).

![Figure 5.32: Keyword matching](image)

On combining KW1 with KW2, KW1 with KW3 and KW1 with KW4 all the different sets of pairs may indicate for different keywords as follows.

\[
\begin{align*}
KW1 + KW2 &= X \\
KW1 + KW3 &= Y \\
KW1 + KW4 &= Z \\
KW2 + KW3 &= W \\
KW2 + KW4 &= U \\
KW3 + KW4 &= V
\end{align*}
\]
Permutations and combinations are used to determine the number of ways in which it's possible to select or arrange a group of keywords. If a user has a string of keywords, say KW1, KW2, and KW3, the possible ways to arrange these are KW1KW2KW3, KW1KW3KW2, KW2KW1KW3, KW2KW3KW1, KW3KW1KW2 and KW3KW2KW1. In each case, the keywords are arranged in a different order, and the total number of different possible ways is six. This is called a permutation. There is another term that is also used when talking about ways of selecting objects from a set, i.e. combinations. The difference between the two is that in permutations, the order of the objects is important. Here the order in which the objects appear is ignored and combine cases where the order is different into a single case.

For example,

```
Believes → in → God → Theist
```
```
Who → Eats → human → flesh → Cannibalism
```
```
Knows → everything → Omniscience
```

In Permutations of n objects, taken r at a time; when repetition is not allowed, the total number of permutations are \( \frac{n!}{(n-r)!} \), whereas if repetition is allowed the permutations are \( n^r \). For combinations of n objects, taken r at a time, the total number of combinations when repetition is not allowed are \( \frac{n!}{[r! \times (n-r)!]} \), whereas if repetition is allowed the number of combinations are \( \frac{(n+r-1)!}{[r!(n-1)!]} \). This shows that for a set of three keywords, the different combinations may be, KW1, KW2, KW3, KW1+KW2, KW1+KW3, KW2+KW3, KW1+KW2+KW3. The total number of combinations generated will be \( 3!/1!*2!] + 3!/2!*1!] + 3!/3!*0!] = 3 + 3 + 1 = 7. 

A migrating crawling agent roams around the web world and executes on a remote platform. Because of mobility of crawling agent, the security problems have become a bottleneck for development and maintenance of mobile agent technology. So, there is a need to develop secured mobile agents and to firm upon certain issues like, maintaining integrity of the network, to be protected from the attacks of other agents and remote...
platforms. The next chapter presents a reliability based approach that is helpful in maintaining security and integrity of mobile agents as well as for data it carries and the remote platform.