CHAPTER 3
INTELLIGENT AGENTS IN TASK ALLOCATION AND DYNAMIC LOAD BALANCING

The performance of a parallel application on a distributed environment is mainly dependent on the allocation of tasks to the geographically distributed workstations to the available processors in the systems. For an effective task allocation in a parallel and distributed environment, this research work uses the following steps with the intelligent agents.

i. The CPU idle time is retrieved from the distributed workstations.

ii. To perform the task allocation process across distributed workstations with task allocation algorithms.

iii. To monitor the distributed workstations when performing the mining process.

iv. To perform the dynamic remote memory utilization.

v. To integrate the knowledge from distributed workstations.

The CPU idle time is retrieved from the identified distributed workstations in a network. Based on this idle time, the various task allocation process is carried out. The workstation which is highly idle will be allocated with the maximum tasks. As task allocation is a major issue in distributed environment, this process is carried out based on the idle time and a task allocation algorithm using intelligent agents.
Intelligent agents perform the allocation process based on a particular task allocation algorithm. A large number of task assignment algorithms have been proposed using various techniques such as network flow, state space search, clustering, bin packing and probabilistic and randomized optimization [MI, 98]. Most of the earlier work of task assignment in Distributed Computing Systems by Harold Stone and Bokhari includes network flow algorithms on dual processor system only; later on researchers took more than two processors into consideration. The task assignment problem solving approaches can be roughly classified into four categories, graph theoretical, mathematical programming, heuristic techniques and probabilistic approaches such as simulated annealing based techniques, mean yield annealing and genetic algorithms [SS, 08].

3.1 TASK ALLOCATION

Task allocation is the process of partitioning a set of jobs into a number of processing groups where each group executes on a separate works. The manner in which this partitioning and allocation is made by determining the efficiency of a given application when it is executed on distributed environments. If this step is not performed properly, an increase in the number of processors may actually result in a decrease in the total system throughput. This degradation is called the “saturation effect”. Hence this is taken care to obtain better system throughput.

A partial taxonomy of well known scheduling methods is presented in Figure 3a. A similar classification is found in the works of Casavant and
Kuhl [Casavant and Kuhl, 1988] and Kwok and Ahmad [Kwok and Ahmad, 1999b].

![Diagram of scheduling methods]

Figure 3a: A partial taxonomy of methods for scheduling concurrent applications to multiple processors.

At the highest level, the taxonomy distinguishes between two categories of methods: (a) job scheduling, and (b) allocation and scheduling. Job scheduling methods are relevant in the context of regulating operations in factories and assembly lines. The goal is to distribute independent jobs or tasks across multiple processing elements to optimize system performance.

In this research work, simulated annealing algorithm is used for task allocation. Simulated annealing is also a global optimization technique that generalizes the concept of Markov Chain Monte Carlo. The basic idea is to iteratively perturb a system until it reaches a state with globally minimum energy. An objective, or cost function, is used to measure the quality of
some property of the system at any state. Each step of the algorithm considers a move from the current state to a random nearby state. The move is retained if the new state improves the evaluation of the cost function; otherwise it is retained with a probability that depends on the difference of quality between the states and a global parameter, called "temperature", which is gradually decreased as the algorithm progresses. Simulated annealing is really a meta-algorithm: the state transition probabilities and the schedule for decreasing temperature are geared to guide the algorithm to reach states quickly which represents good solutions for a specific optimization problem.

Simulated annealing has found successful application for multiprocessor scheduling problems. The method determines proper terminal temperature settings and the number of necessary iterations per temperature to ensure rapid convergence. It can also flexibly incorporate varied implementation and resource constraints.

3.1.1 Task Allocation Algorithm - Simulated Annealing (SA)

In a parallel and distributed environment, task allocation is a major activity performed for allocating the job to different distributed locations based on the algorithm. This section presents a heuristic algorithm derived from the well known SA technique [GY, 06]. The algorithm starts by randomly selecting an initial solution \( s \) and computes the cost \( E_s \) at the current solution \( s \). After setting an initial temperature \( T \), a neighbor finding strategy is invoked to generate a neighbor solution \( n \) to the current solution
s and compute the corresponding cost $E_n$. If the cost $E_n$ at the neighbor solution $n$ is lower than the current cost $E_s$, then the neighbor solution is accepted as a current solution. Otherwise, a probability function $\exp(-\Delta/T)$ is evaluated to determine whether the neighbor solution may be accepted as a current solution, where $\Delta = E_n - E_s$. After a thermal equilibrium is reached at the current temperature $T$, the value of $T$ is decreased by a factor $a$ and the number of inner repetitions is increased by an increasing factor $\beta$. The algorithm continues from the current solution point searching for a thermal equilibrium at the new temperature level. The whole process terminates when either the lowest cost point is found or no upward/downward jumps have been taken for a number of successive thermal equilibrium. The structure of the algorithm may be sketched as follows:
Randomly select an initial solution s;  
Compute the cost at this solution Es;  
Select an initial temperature T;  
Select a cooling factor a < 1;  
Select an initial chain \( n_{rep} \);  
Select a chain increasing factor \( \beta > 1 \);  
Repeat  
Repeat  
Select a neighbor solution \( n \) to \( s \);  
Compute the cost at \( n \), \( E_n \);  
\( \Delta = E_n - E_s \);  
If \( \Delta < 0 \),  
\( s = n \); \( E_s = E_n \);  
Else  
Generate a random value \( x \) in the range \((0, 1)\);  
If \( x < \exp\left(-\frac{\Delta}{T}\right) \),  
\( s = n \); \( E_s = E_n \);  
End  
End  
Until iteration = \( n_{rep} \) (equilibrium state at \( T \))  
Set \( T = a \times T \);  
Set \( n_{rep} = \beta \times n_{rep} \);  
Until stopping condition = true  
Es is the cost and \( s \) is the solution.

Figure 3b: Simulated Annealing Algorithm

This algorithm penalizes the following characteristics:

i. a task redundancy,

ii. a processor with a load utilization > 100%,

iii. a processor with a memory utilization > 100%,

iv. a link/channel with a communication capacity utilization > 100%.

These characteristics are penalized to achieve the application requirements and validate the availability of the system resources. In this
research work, the first property is penalized by constructing an allocation vector $A(M, 1)$ whose element $A(i)$ represents the processor $p$ where the task $i$ is allocated. At each movement of neighboring solutions, one of the tasks is moved from one processor to another. Therefore, each task cannot be allocated to more than one processor. The second property is penalized by comparing the processing load requirements of all the tasks mapped to a processor $p$ and the available processing load of $p$. A component $Ep$ is determined such that $Ep = 1$ if the processing load requirements exceed the available load of $p$ and $Ep = 0$ otherwise. Similarly, the third and the fourth properties are penalized. The third property is penalized by returning a cost component $Em$ such that $Em = 1$ if the memory requirements of all the tasks mapped to a processor $p$ exceed the available memory at $p$ and $Em = 0$ otherwise. Also, the fourth property is penalized by returning a cost component $Ec$ such that $Ec = 1$ if the communication capacity requirements of all edges mapped to a link/path $pq$ exceed the available capacity of the path and $Ec = 0$ otherwise [GY, 06]. Let $k$ be a penalty factor, then the function $E$ may be formulated as follows:

$$E = Z + k(E_p + E_m + E_c).$$

To implement the SA algorithm, a number of decisions are made. These decisions are concerned with the choice of an energy function, a cooling schedule, a neighborhood structure and the choice of annealing parameters such as an initial temperature, a cooling factor, an increasing factor of the inner loop repetitions and a stopping condition. Each decision is made with care as they affect the speed of the algorithm and the quality of solutions.
This research work uses a heuristic algorithm for task allocation well suited in a distributed environment, namely the simulated annealing algorithm. The different task allocation parameters identified in this research work for allocating the job as follows.

A set of $M$ tasks denotes a parallel application to be executed on a distributed system of $N$ processors. Tasks of the given application require certain computer resources such as computational load and memory capacity which communicates at a given rate. There is a failure rate with each component in the system resources. Here $M$ tasks have to be allocated to one of the $N$ processors such that the overall system reliability is maximized, the requirements of tasks and edges are met, and the capacities of the system resources are not violated. In a distributed system, each component exists in one of two states: operational or faulty [GY, 06]. In order for a parallel application to run successfully on a distributed system, each processor must be operational during the time of processing its tasks and each path must be operational during the active period of data communication between the terminal processors of the path. Hence, successful execution of the application is mainly dependent on reliability of the system components (processors and links) and on the distribution of the application tasks to the available processors in the system. To do so, let $X$ be an $M \times N$ binary matrix corresponding to an assignment of $M$ tasks onto $N$ processors such that an element $X_{ip} = 1$ if a task $i$ is assigned to a processor $p$ and $X_{ip} = 0$ otherwise. [GY, 06].
3.1.2 Processor Reliability

Processor reliability is the first parameter identified in this research for task allocation to find out the reliability of the processors by allocating each processor with its respective task without overloading a single processor. Processor reliability \( R_p \) is the probability that the processor \( p \) is operational for execution of tasks that are assigned to it. Define \( \lambda_p \) as the failure rate of a processor \( p \) during a time interval \( t \), then the reliability of the processor \( p \) in the time \( t \) is \( e^{-\lambda pt} \) [Hsi, 03 GY, 06]. Under a task assignment \( X \), the time \( t \) represents the time required to execute all the tasks assigned to \( p \). Let \( C_{ip} \) be the time of processing a task \( t \) on a processor \( p \), then the processor reliability may be formulated as

\[
R_p = e^{-\lambda_p \Sigma C_{ip} X_p}
\]

The summation gives the total time elapsed in executing all the tasks that are assigned to the processor \( p \).

3.1.3 Path Reliability

The path reliability is the second parameter considered for communication path between workstations which is an important parameter in task allocation in which the number of workstations to determine the possible route. Path reliability \( R_{pq} \) is the probability that the path \( pq \) is operational for communicating data between the terminal processors of the path. A path is sequence of communication links from a sender to a receiver. Define \( \lambda_{pq} \) as the failure rate of a path \( pq \) during a time interval \( t \), then the reliability of the path \( pq \) in the time \( t \) is \( e^{-\lambda_{pq} t} \) [Hsi 03, GY 06]. Under a task
assignment $X$, the time $t$ represents the interval required for data communication between the terminal processors of the path $pq$. Let $C_{i\|pq}$ be the time of transferring data between tasks $i$ and $j$ if they are assigned to different processors $p$ and $q$, then the path reliability may be formulated as

$$R_{pq} = e^{-1pq} \sum_{i \neq j} C_{i\|pq} X_{ip} X_{jq}.$$  

The summation gives the total time required for communicating data between the terminal processors ($p$ and $q$) of the path $pq$. From the components reliabilities, it is clear that, allocating tasks of large execution times to more reliable processors is a good approach to increase the execution reliability in the system. Also, allocating edges of higher communication times to more reliable links is a good approach to increase the communication reliability in the system [GY, 06].

3.1.4 System Reliability

The third parameter considered for task allocation in this research work is the system reliability. A system cannot be deemed reliable if it is not secure, so the concepts of security and reliability are intrinsically linked. System reliability of a distributed system $R_{sys}$ may be defined as the probability that the system can run the entire application successfully [RN, 95]. In other words, the system reliability is the product of the components reliabilities. That is, the product of the probability that each processor be operational during the period of tasks execution and the probability that each path/link be operational during the period of interprocessor communication.
Distributed system architecture consists of a collection of \( K \) processors with distributed memory, i.e. with sufficient memory at each processor so that any one task can be executed. The processors are fully interconnected by a high speed network. A parallel program is executed in this environment denoted by a task graph [JE, 01].

\[ \Pi = (N, A, e, C), \]

Where \( N = \{1, \ldots, n\} \) is the set of \( n \) tasks that compose the program,

\[ A = \{a_{ij}\} \]

is the incidence matrix which describes the graph,

and \( e, C \) are the amount of work related to task execution and to communication between tasks.

\( C_{ij} \) denotes the amount of information transferred during communication from task \( i \) to task \( j \).

If \( a_{ij} = 1 \). Clearly \( a_{ij} = 0 \) implies that \( C_{ij} = 0 \).

The task assignment is assigning the \( n \) tasks to \( K \) processors. This means that partitions are found \((\Pi_1, \ldots, \Pi_k)\) of the set of \( n \) tasks which optimizes the performance, as expressed by following criteria

- The communication between different processors of the system must be kept to a minimum.
- The load of the different processors must be balanced.
- The total effective execution time of the parallel program must be minimized.

Task assignment usually refers to decisions made before program execution. This approach to distributing processes or tasks to processing units can be applied effectively to programs whose run-time behavior is
relatively predictable, since the decision must rely on a prior knowledge of
the system obtained from the CPU idle time.

Task allocation is carried out so as to optimize a criterion which
describes the "costs". The assignment is carried out by a set of binary
variables \( \{ X_{ip} \} \) where \( i \) range over the set of tasks and \( p \) ranges over the set of
processors. Thus \( X_{ip} \) is the binary variable whose value is 1 if task \( i \) is
assigned to processor \( p \) and 0 otherwise.

\[ X_{ip} \cdot X_{jq} = 0 \, \text{for all } i, \ p \neq q \, , \ \text{and } \sum_p x_{ip} = 1 \, \text{for all } i. \]

The main components of the cost function are given below

- **The Total Program Execution Time**

  This cost depends on the "size" of the tasks, expressed in size of
  executed code, or in execution time on some normalized processor,
  and on the speed of the processors. It can be expressed as

  \[ CE = \sum_p U_p X_{ip} \]

  where

  - \( e_i \) is the size of task \( i \) in number of instructions executed,
  - \( U_p \) is the time of execution for one instruction on
    processor \( p \)

- **The Communication cost**

  This is one of the important factors to be minimized by the task
  assignment. It depends on the quantity of information to be
  exchanged between tasks.

  \[ C_C = \sum_p \sum_{l \neq p} \sum_{j \neq l} \left( CCP_{pjql} + CCP_{qljl} + C_{jl}D_{pq} + C_{lj}D_{qp} \right) X_{lp} X_{jq} \]
where

- \( C_{P_{ij}} \) is the time necessary for processor \( p \) to set up the communication between tasks \( i \) and \( j \).
- \( C_{ij} \) is the total quantity of information transferred between tasks \( i \) and \( j \).
- \( D_{pq} \) is the average time needed to transfer a unit of information from processor \( p \) to processor \( q \).

- **The cost of Access to the Files**

The cost of access depends on the task situated and the time it will take to access files also varies. A file \( f \) will be resident on processor \( i \) if the binary variable \( Y_{ip} \) takes value 1 otherwise it will be 0.

\[
C_F = \sum_p \sum_{i \neq \phi} \sum_{f} (CCF_{pi\phi} + CCF_{qfp} + V_{ip}D_{pq}) X_{ip}Y_{iq}
\]

where

- \( V_{if} \) is the average quantity of information that task \( i \) needs from file \( f \).
- \( CCF_{pif} \) (\( CCF_{qfp} \)) is the average time necessary for processor \( p \) to set up communication between task \( i \) on processor \( q \) and file \( f \) on \( q \).

The simulated annealing algorithm is applied to provide good approximations with optimal solution.
3.2 Dynamic Load Balancing

After performing the task allocation process, the mining process is carried out in the distributed workstations in parallel. The intelligent agent monitors the complete mining process so that if there is any system crash or any fault in the workstations the agents act accordingly by providing the dynamic remote memory utilization.

Load balancing is the assignment of work to processors which will be critical in parallel simulations. It maximizes application performance by keeping processor idle time and interprocessor communication as low as possible. In applications with constant workloads, static load balancing can be used as a pre-processor to the computation. Other applications, such as adaptive finite element methods, have workloads that are unpredictable or change during the computation; such applications require dynamic load balancers that adjust the decomposition as the computation proceeds. Numerous strategies for static and dynamic load balancing have been developed, including recursive bisection (RB) methods, [BB, 87], [Sim, 91], [TN, 94], space-filling curve (SFC) partitioning [Mit, 95], [FSS+, 98] and graph partitioning.

As association rules are used in the mining process, it may happen that the number of candidate itemsets increases dramatically in this step so that the requirement of memory becomes extremely large. When the required memory is larger than the real memory size, part of the memory contents must be swapped out [BGL, 07]. The number of candidate item sets in pass 2 is very much larger than other passes in association rule mining.
The number of item sets is strongly dependent on user-specified conditions, such as minimum support value, and it is difficult to predict before execution how large the number will be before execution.

In this research work the remote workstations, whose memories are available for application execution workstations, are found dynamically during the execution. It is called as "memory available workstations". On memory available workstations, a process is running to monitor the amount of available memory periodically. Each time the process gets the information, the process broadcasts it to all application execution workstations. On application execution workstations, a client process is running and waiting for the information sent from the memory monitoring processes running on memory available workstations. The client process has a memory area which can be shared with application processes, and the received information about the amount of memory at each workstation is written on the shared memory. The application processes can read this information at anytime, to decide the policy of swap in and swap out operations. For example, when a memory available workstation does not have enough memory space, the shortage of memory is detected by application processes, so that another workstation is chosen as a swapping destination afterward.

On the other hand, if some other processes begin their execution on a memory available workstation which is already accepted with swap in and swap out operations, the swapped out data must be migrated to other
memory available workstation to make space on its memory for the new processes.

A Data warehouse D contains different data sets like \( \{D_1, D_2, D_3, \ldots, D_n\} \) in a geographically distributed environment

\[ \{D_1, D_2, D_3, \ldots, D_n\} \subset D \]

where these datasets are assigned to the different processors \( \{P_1, P_2, P_3 \ldots, P_n\} \). The dataset assigned to these processors may vary dynamically depending upon the mining process carried out using Hash Partitioned Apriori. If there is any processor idle, then the load will be properly balanced to every processor, so that the communication overhead will be reduced.

Initially, a method using available remote workstations memory with simple swapping is examined. The maximum number of workstations used as memory available workstations is changed from 1 to 16. In this research work, all memory available workstations are assumed to have enough memory space to accept requests of swapping operations. In such a case, all the memory available workstations are used for swap operations throughout the execution of the program and therefore the number of memory available workstations is constant during the experiment. The parameters considered for this research in dynamic remote memory utilization are as follows.

**3.2.1 Execution Time**

The time taken for executing the process is analyzed. The scalability of the parallel applications by executing them on 1, 4, 8 and 16 processors is measured. The total execution time and the communication delay are
represented graphically. It is found out that the execution time is minimized by dynamic load balancing.

3.2.2 I/O Time

In general, I/O is a key component that could affect the overall performance of a system. The time for performing I/O as a percentage of the overall execution is studied. It is found out that the time taken in execution is minimized due to the dynamic load distribution.

3.2.3 Candidate Migration

This process is used to balance the load in parallel association rule mining. While using Hash Partitioned Apriori, each workstation receives the itemsets and probes them against its own hash table. If a workstation is assigned more candidate item-sets and keeps them as a hash table, it will receive more itemsets from other workstations during counting phase. This means that the workload of each workstation is adjusted by adjusting the amount of candidate itemsets. If the load of a certain workstation is higher than the other workstations, some of the candidate itemsets are taken from that workstation and give them to the other workstations. Then the itemsets that are originally directed to that workstation are now redirected to the new workstations to which the removed itemsets are relocated. Thus the counting workload is migrated from the original workstation to the other workstations. This strategy is named as Candidate Migration [MM,01].

76
The workload depends on the itemset. Some item-sets have higher support value, which means those itemsets will receive more counting requests. Thus weighting factor is given to each itemset. However real support value is obtained after the execution.

Support counting for pass k in HPA can be divided into 2 processes. The first process is SEND process which comprises reading transaction data from disks, generating k-itemsets and sending the k-itemsets to destination workstations by applying hash function to the itemsets. The second process is called RECV process since processing workstation receives the k-itemsets from other workstations then probes its hash table to increase the support counts of candidate itemsets. If the time required by workstation i for SEND process and RECV process are expressed by $ST_i$ and $RT_i$ respectively, overall CPU processing time for that workstation can be formulated by:

$$\Delta T_i = \Delta ST_i + \Delta RT_i$$

(1)

Here $\Delta$ stands for definite time interval.
If the required time is estimated for each workstation’s SEND and RECV processes to complete the processing of remaining transaction data, the estimated remaining processing time for that workstation is obtained as indicated by following expression.

\[\text{restT}_i = \text{restST}_i + \text{restRT}_i\]  \hspace{0.5cm} (2)

Each of restT, restST, and restRT denotes estimated remaining overall CPU processing time, estimated remaining CPU time for the SEND process and estimated remaining CPU time for RECV process respectively.

The main goal is to have all the workstations to complete their job more or less at the same time, this method dynamically controls the load allocated for each workstation so that every workstation has the same restT.

The skew is given by,

\[\text{skew} = \frac{\text{max}(\text{restT}_i) - \text{min}(\text{restT}_i)}{\text{avg}(\text{restT}_i)}\]  \hspace{0.5cm} (3)

\[
\begin{align*}
\text{skew} \leq \text{threshold} & : \text{no skew} \\
\text{skew} > \text{threshold} & : \text{skew exists}
\end{align*}
\]  \hspace{0.5cm} (4)

RECV process can be divided further into sub processes: (recv1) receives k-itemsets and (recv2) probes the hash table and increment the support count for the corresponding candidate itemsets. If the time for each subprocess is represented by RNT and RCT respectively, RT can be expressed as follow:

\[\Delta\text{RT}_i = \Delta\text{RNT}_i + \Delta\text{RCT}_i\]  \hspace{0.5cm} (5)
Since large scale data mining has to probe large amount of candidate itemsets against hash table, most part of processing time is dominated by (recv2). Thus expression (5) can be reduced to:

\[ \Delta RT_i = \Delta RCT_i \]  

(6)

RCT_i itself is proportional to the amount of k-itemsets to receive, thus if that amount is expressed as RK_i, assume:

\[ \Delta RCT_i = \alpha_i \Delta RK_i \]  

(7)

RK_i varies according to the candidate itemsets allocated to that workstation. RK_i is unknown before execution.

If the support count for an itemset X is defined as SX, a weighting factor for that candidate itemset is as follows:

\[ CW_{cand_j} = \min\{S_{l_{j1}}, S_{l_{j2}}, \ldots, S_{l_{jk}}\} \]  

\[ \{L_{j1}, L_{j2}, \ldots, L_{jk}\} \in \text{Sub}_j \]  

(8)

Sub_j denotes a set of all large (k-1)-itemsets that are subsets of cand_j.

RK_i represents a set of all candidate itemsets in pass k. Then if CD_i is represented as a set of candidate itemsets to be allocated to workstation i, the weighting factor for that workstation can be defined as:

\[ CV_i = \sum_{j}^{CD_i} CW_{ca} \]  

(9)

cand_j \in CD_i

If overall amount of transactions to be read by all workstations is expressed as DR, RK_i is proportional to the product of CV_i and DR. Therefore, expression (7) is expanded further into:
\[ \Delta RCT_i = a_i^1 \ a_i^2 \ CV_i \text{restDR} \] 

(10)

From expression (2) rest\(T_i\) is expressed as

\[ \text{rest}T_i = \text{rest}ST_i + a_i^1 \ a_i^2 \ CV_i \text{restDR} \] 

(11)

SEND process performs the following subprocesses (send1) read transaction from data warehouse (send2) generates k-itemsets (send3) send them to proper workstations.

If the time for each subprocesses is denoted as SDT\(_i\), SCT\(_i\), SNT\(_i\), respectively, ST\(_i\) is expressed as

\[ \Delta ST_i = \Delta SDT_i + \Delta SCT_i + \Delta SNT_i \] 

(12)

\[ \Delta ST_i = \Delta SCT_i \] 

(13)

Since the load for (send2) subprocess depends on the amount of k-itemsets to send and this amount is proportional to DR\(_i\) that is the amount of transactions to be processed by workstation \(i\),

\[ \Delta SCT_i = a_i^3 \Delta DR_i \] 

(14)

Hence expression 2 is expanded as

\[ \text{rest}T_i = a_i^3 \text{restSK}_i + a_i^1 \ a_i^2 \ CV_i \text{restDR} \] 

(15)

SK\(_i\) denotes the amount of k-itemsets for workstation \(i\) to send out.

Candidate migration requires the remapping of hash table. Once the itemset is migrated, new destination address is put onto the entry so that itemsets are appropriately distributed over the workstations. If remapping is made itemset by itemset, it costs a lot of space.
3.2.4 Transaction Migration

When load is extremely skewed, the workload migration of RECV process cannot be relied on. Thus, another load balancing strategy based on the SEND process is used. As described before, in most cases of large scale data mining, processing time is dominated by (send2). The load for (send2) sub process depends on the amount of k-itemsets to be sent, this can be expressed as SCTi.

\[ \Delta SCT_i = \Delta SK_i \]

Here SKi is defined as the amount of k-itemsets for workstation i to send. Each workstation in PC cluster has its own partition of transaction data, Transaction Migration sends some part of that transaction data to other workstations and delegates the generation of k-itemsets to those workstations. In order to do this, each workstation holds a list of destination workstations and their assignment. This list is dynamically updated during execution. This approach can remove the burden of SEND process of heavy workstations to workstations with excessive computing power, thus it can effectively balances the workload. As mentioned before, the amount of k-itemsets sent to other workstations SKi is proportional to the amount of k-itemsets generated by that workstation. Therefore if workstation j delegates DMj (= -DMj) k-itemsets generation to workstation i and the ratio of DMj against all the transaction data in workstation j is defined as TMij(= -TMij). Then SKi can also be expressed as follows [MM,01]:

81
\[ \Delta S K_i = (\Delta D R_i + \sum_{j,j' \neq i} \Delta D M_{ij}) \]

\[ \Delta D M_{ij} = \{ T M_{ij} \Delta D R_j \mid T M_{ij} > 0 ; T M_{ij} \Delta D R_i \mid T M_{ij} > 0 \} \]

Then will the previous equation will be turned to

\[ \Delta S C T_i = f(\Delta D R_i + \sum_{j,j' \neq i} \Delta D M_{ij}) = f(\Delta D R_i + \sum_{j,j' \neq i} \Delta D M_{ij} > 0 \sum_{j,j' \neq i} \Delta D R_j) + \sum_{j,j' \neq i} \sum_{M_{ij} < 0} T M_{ij} \Delta I \]

\[ \text{rest} T_i = f(\text{rest} D R_i + \sum_{j,j' \neq i} \sum_{T M_{ij} > 0} T M_{ij} \text{rest} I) + \sum_{j,j' \neq i} \sum_{T M_{ij} < 0} T M_{ij} \Delta I + \alpha_1 \alpha_2 \text{CV}_{\text{rest} D R} \]

From the above equation \( \text{rest} T_i \) can be controlled by delegating \( k \)-itemsets generation of SEND process. If Candidate migration is not enough to overcome skew, this can be supplemented with Transaction Candidate.

Coordinator sends migration plan to all processing workstations and instructs them to reallocate the load. When Candidate Migration is employed, each workstation transfers candidate itemsets according to the plan and renews the hash table. It also remakes the destination list of transaction data if Transaction Migration is needed. HPA (Hash Partitioned Apriori) algorithm is adopted for underlying parallel association rule mining. This partitions the candidate itemsets over the workstations, while ordinary methods just copy candidate itemsets all over the workstations. HPA can improve the memory efficiency significantly. Two kinds of dynamic load balancing strategies for parallel association rule mining are used, Candidate Migration and Transaction Migration. By changing the allocation of candidate itemsets among the workstations, the workload of the
workstations can be identified. The amount of candidates to be migrated can be derived analytically.

3.3 INTELLIGENT AGENTS

Agents are defined as software or hardware entities that perform some set of tasks on behalf of users with some degree of autonomy [RN, 95]. In order to work for somebody as an assistant, an agent has to include a certain amount of intelligence, which is the ability to choose among various courses of action, plan, communicate, adapt to changes in the environment, and learn from experience. In general, an intelligent agent can be described as consisting of a sensing element that can receive events, a recognizer or classifier that determines which event occurred, a set of logic ranging from hard-coded programs to rule-based inferencing, and a mechanism for taking action [DAA, 95]. An agent is mobile if it can navigate through a network and perform tasks on remote machines. A learning agent adapts to the requirements of its user and automatically changes its behavior to environmental changes. The relationship among agents, architectures, and programs can be summed up as follows:

agent = architecture + program

The intelligent agents used here are mainly for monitoring the data allocation process on a distributed environment, during the mining process and also while integrating the knowledge.
The *agent function* is a mathematical function that maps a sequence of perceptions into action.

- The function is implemented as the *agent program*.
- The part of the agent taking an action is called an *actuator*.

\[
\text{Environment} \rightarrow \text{sensors} \rightarrow \text{agent function} \rightarrow \text{actuators} \rightarrow \text{environment}
\]

### 3.4 TYPES OF AGENTS

Several categories or types of agents have been defined, based on their abilities and, more often on the task they are designed to perform. The difference is their capabilities for doing that task. It seems that with intelligent agents, as with people, knowledge and adaptability is differentiating the successful ones from the less effective ones. The major categories recognized for this research work are filtering agents, information agents, user interface agents, office or workflow agents, system agents [Ays, 99], brokering or commercial agents.
3.4.1 Filtering Agents

Filtering agents, as their name implies, act as a filter that allows information of particular interest or relevance to users to get through, while eliminating the flow of useless or irrelevant information. The filtering agents work in several ways, in which the user provides a template or profile of the topics or subjects that are of interest. When presented with a list of documents in a database, a filtering agent scans the documents and ranks them based on how well their content matches the user's area of interest. In this research work, as data is allocated to different geographical locations which may have historical data, but depending on the necessity filtering agents are used to filter the relevant information and perform the mining process.

**Developed Filtering Agent Algorithm**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Filtering Agent is initialized</td>
</tr>
<tr>
<td>2</td>
<td>Data warehouse with historical data is filtered based on the condition.</td>
</tr>
<tr>
<td>3</td>
<td>The filtered data is then distributed to the different workstations.</td>
</tr>
<tr>
<td>4</td>
<td>Mining process starts</td>
</tr>
<tr>
<td>5</td>
<td>Stop</td>
</tr>
</tbody>
</table>
3.4.2 Information Agents

Information agents which are used primarily on the internet and worldwide web can scan through online databases, document libraries, or through directories in search of documents that might be of interest to the user. In this research work with distributed data mining the information agents can update the online database automatically from time to time to perform the mining process.

**Developed Information Agent Algorithm**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Information Agent is initialized</td>
</tr>
<tr>
<td>2</td>
<td>Data warehouse with historical data is present.</td>
</tr>
<tr>
<td>3</td>
<td>The information agents perform the database updating process.</td>
</tr>
<tr>
<td>4</td>
<td>Mining process starts</td>
</tr>
<tr>
<td>5</td>
<td>Stop</td>
</tr>
</tbody>
</table>

3.4.3 User Interface agents

User interface agents monitor the user interactions with the application and can control various aspects of that interaction, such as the level of prompting or the number of options available. For example in this research as data is dealt with from a distributed environment there is a vital role of user interface agents. The user interface agents make the system more user-friendly by having the required training explicitly given by the user.
Developed User Interface Agent Algorithm

Step 1: The User Interface Agent is initialized and activated.
Step 2: The agent monitors the task allocation process.
Step 3: The user interface agents monitors the memory allocation.
Step 4: Stop

3.4.4 Office or Workflow Agents

An office management agent automates kinds of routine, daily tasks that take up so much time at the office. These tasks include scheduling meetings, sending faxes, holding meeting review information, and updating process documents. In a parallel and distributed data mining process the workflow agents define the path in which the mining is carried out after an effective task allocation.

3.4.5 System Agents

System agents are software agents whose main job is to manage the operations of a data communications network. These agents monitor for device failures or system overloads and redirects work to other parts in order to maintain a level of performance or reliability. In a parallel and distributed data mining environment system agents are used to locate the different workstations and used mainly in dynamic remote memory utilization where the remote memory is utilized for the mining process.
Developed Sysems Agent Algorithm

Step : 1 System Agent is initialized
Step : 2 System agent monitors the distributed workstations.
Step : 3 System agent monitors dynamic remote memory utilization.
Step : 4 Also monitors during mining process.
Step : 5 Stop

3.4.6 Mobile Agents

An agent needs to interact with its host system and other agents in order to be useful. Mobile agents are defined as programs which may be dispatched from a client computer and transported to a remote server computer for execution and interaction with other agents [HCK, 95]. In this research work mobile agents are used to locate the distributed workstations and retrieve the idle time of the systems.

Developed Mobile Agent Algorithm

Step : 1 Mobile Agent is initialized
Step : 2 Mobile agents monitor the distributed workstations.
Step : 3 Mobile agents are all time mobile to monitor the mining process.
Step : 4 Stop
These are the reasons indicated for using mobile agents in [LO, 99], which are:

- reducing the network load.
- overcoming network latency in real-time systems.
- Asynchronous and autonomous execution (dispatching from the mobile devices and becoming independent),
- Dynamic adaptation (sense and adapt to changes, optimal configuration for solving a problem),
- Heterogeneity (dependent only their execution environments),
- Robust and fault tolerant (dynamic, continue on another host).

### 3.5 Performance measures in intelligent agents

\[ IAP = w_1 \text{ accuracy} + w_2 \text{ Speed} + w_3 \text{ Benefit} + w_4 \text{ Cost} \]

The idea of intelligent software agents has captured the popular imagination. The agent is instructed on what process is to be carried out, set it free, and waits for it to return results. Wooldridge and Jennings (1995) define an intelligent agent as one that is capable of flexible autonomous action to meet its design objectives.

The accuracy of the agent is an important parameter because it involves in monitoring of the workstations before and during the mining process. Hence it must be accurate to provide optimized result. The intelligent agents perform all tasks in faster rates so that high speed outcomes could be obtained.
The agent plays the task of dynamic remote memory utilization so that the benefit of the memory utilization is achieved. The memory overload problem is minimized. The swap in swap out processes is carried out effectively. The time taken for communication is minimized because agents are involved in performing the task effectively.

In this research work the process of task allocation is carried out with the simulated annealing algorithm along with intelligent agents. The Dynamic load balancing is taken care by various parameters discussed above so that the memory is utilized effectively by obtaining a better system throughput.