# The Optimal Resource Discovery and Allocation Algorithm for One hop Using Anonymous Arbitrary Topology

Mr.Srinivasan Nagaraj Mr.K.Koteswara Rao Mr.G Appa Rao Mr.TV Madhasudanarao Dr.GSVP Raju
**GMR Inst. Of Tech.** **GMR Inst. Of Tech.** **GITAM University** **TPIST** **AndhraUniv**
RAJAM-532127,India RAJAM-532127,India Vizag-530041India Bobbili,India Vizag-530041,India

## ABSTRACT
A distributed system consists of, possibly heterogeneous, computing nodes connected by communication network that do not share memory or clock. One of the main benefits of distributed systems is resource sharing which speeds up computation, enhances data availability and reliability. However resources must be discovered and allocated before they can be shared. ***Virtual caching*** is a new caching scheme which allows a host node to grant authority of caching pages in some fraction of its own cache to nearby nodes. However the virtual caching protocol doesn't mentions how a client node obtains virtual cache from remote host. To address this problem we formulate a resource discovery and allocation problem. We are focusing our attention on how to locate resources-surplus ***donor*** nodes and to determine how much of the request for resources of ***deficient*** nodes will be satisfied, efficiently in a connected network especially within a finite hop of the resource deficient node. We intend to minimize the amount of unfulfilled request of deficient nodes. Virtual cache allocation can be changed any time depending upon the requirement. Hence the proposed heuristics are efficient both in terms of time and amount of communication performed.
We also estimate the quality of distribution achieved by comparing the distribution yielded by the heuristics and by the solution of ILP formulation of the problem. We propose and compare few heuristics for minimizing the amount of unfulfilled request for resources, of deficient nodes when nodes look for resources within finite hops. In this paper we are restricting ourselves to single hop only. For the bounded hops we restrict ourselves to the resource distribution within one hop. By using non-anonymous arbitrary topology with sequence number of request to resolve deadlocks and distributing resources over the original arbitrary network. Sequence number of the request is the unique ID of sender node. We proposed a heuristic to distribute resources over anonymous arbitrary topology by passing a token . The token is privilege to distribute the resources. Each resource - surplus node is giving its extra nodes in such a way so that it itself doesn't becomes resource-deficient in the process. Load is not infinitely divisible. We are focusing our attention only to determine how much of the request of each resource - deficient node will be satisfied.
**Keywords:** donor, deficient, token, caching, optimal

## Chapter 1

### *1.1 Introduction*
A distributed system is a collection of loosely coupled processors interconnected by communication network [21]. One of the main advantages of the distribution system is resource sharing. However in order to maximize the resource utilization, they must be discovered and allocated efficiently. To make the resource utilization more efficient caching is used. Virtual memory systems may be viewed as caching secondary storage data into faster main memories. Carrying this principle into distributed systems, Virtual caching was proposed .In Virtual caching allows a host node to grant authority of caching pages in some fraction of its own cache to nearby nodes. Virtual caching may be compared with virtual memory; in either case, there is a larger accessible virtual address space, with mappings into physical spaces outside the physical space of the accessing unit.
*Virtual Caching scheme* is a new caching scheme. In virtual caching scheme, the cache granting node is called the *virtual host* .*Such* a virtual host relinquishes control of some part of its own cache space so that the same can be used by other nodes; we call such other nodes *virtual clients*. *Virtual hosts* can be considered as *donor nodes* which are having surplus resources. *Virtual clients* can be considered as *deficient nodes* which need resources. However the virtual caching protocol doesn't mentions how a. client node obtains virtual cache from remote host. We address this issue by proposing few heuristics and studying their performances.

***1.2 Resource Discovery and Allocation Problem***
The virtual client nodes can be considered as deficient nodes as looking for resources (cache) and virtual host nodes can be considered as donor nodes willing to give resources (cache). Under this paradigm we formulate the resource discovery and allocation problem as follows.

**Problem statement:** There are *n* nodes in a connected undirected network *G=( V,E).* Assume that each edge e =( *i,j)* has an associated non negative real valued weight, *weight(e)=weight;,j'* We assume that for all i and j, *weighf;.,j=weightj,i'* Here weights represent the cost of com-munication between the nodes. Each node ~ is having some capacity (resource owned by the node) Ci and some requirement (resource required by the node) ri. Capacity of each node may be equal or less or greater than the requirement. For sake of simplicity assume that sum of all capacities and requirements over all nodes across the network are greater than or equal; implying that total requirement can be met within the network. Let *R* be set of all the nodes whose requirement of resource is more than their own capacity i.e. *R*= {ni :ri>Ci} Let *S* be the set of all the nodes whose capacity is more than their requirement i.e. *S={* ni : ri<Ci }. Let *T = [tij]* (where *i=l..n* and *j=1..n)* be the transfer matrix denoting the amount of resource which is transferred. *ti,j* < 0 means node ni will receive Ti,j units of resources from node nj. *ti,j* > 0 means node ni will give Ti,j units of resources to node nj.

Task is to devise an efficient algorithm (whose communication complexity is less than $O(N^2)$ ) to minimize the sum,

$\Sigma((ri - Ci) + \Sigma (t_{ij})$ under the constraint that
$ni\varepsilon R$ $nj\varepsilon S$

$\forall$ *ni* E *S*, « Ci - ri) - $\Sigma$ $t_{ij}$ *)* >=0 i.e. we have to satisfy
$\forall$ *njεR*
requirement of resource deficient nodes under the constraint that nodes having surplus resources share resources among resources deficient nodes in such a way that they themselves don't become resource deficient. Also resource deficient nodes can not more resources than their requirement.
We consider two versions of the above problem.

***Problem* 1** *(bounded hops version) PI:* In this version resource-deficient nodes look for resources within finite hops.

***Problem 2****(unbounded hops version) P2:* In this version, we are not limiting the hops within which resource-deficient node can look for the resources.

In order to compare the optimality of our resource distribution algorithm we use the solution given by Integer Linear Programming (ILP) as benchmark. Solution given by (ILP) is the best solution which is possible under the restriction of bounded hops. For the unbounded hops case optimal value of the metric when sufficient resources are
available $\Sigma$ (ri - ci) >=$\Sigma$($r_j$ – $C_j$ *)is 0* otherwise the value of niεS *njεR*

metrics is $\Sigma$ (ri - ci) -$\Sigma$($C_J$ – $R_J$) .
niεS *njεR*

We. compare the value of the metric yielded by these benchmarks to the metric corresponding to our algorithms to find the quality of distribution achieved. We put the constraint that donor nodes give resources in such a way that they don't become resource deficient. Also the deficient nodes don't accept resources more than their requirement. we want to minimize the amount of unfulfilled request of the deficient nodes under these constraints within efficient message and time complexity. We also formulate the resource discovery problem as an optimization problem and solve it using an ILP solver. The solution thus obtained is then compared with the distribution achieved by the heuristics with respect to the amount of unfulfilled request as the metric.

# Chapter 2

## 2.1 Virtual Caching Scheme

### *2.1.1 Schematic of Virtual Caching*

The idea of virtual caching is to get the sole authority to use fraction of the cache space of nearby nodes (called *virtual hosts* or simply *hosts* hereafter) which are presently not utilizing their cache space in full. Such virtual hosts must be able to afford to relinquish
control of the part of its own cache space that is to be used by other node (called *virtual clients* or simply *clients* hereafter )storing data in the cache.

More precisely, virtual cache of a node the virtual cache of a node *A* is cache borrowed by node *A* from some other node *B* in the network. A very active node *A* may reserve (borrow) some part of the cache of some other node *B* which is perhaps not so active and can afford to lend a part of its cache to *A.* This reserved cache is called the virtual cache of node *A* at *B.* Nodes *A* and *B* are called client and host, respectively. Note that the client logically sees a much bigger cache than its physical cache. When a client accesses a page, it first checks in its own physical cache and then in its virtual borrowed caches in other host nodes.

As long as a part of cache space is given to a client by a virtual host, only the client will have the authority to write (store data) to the virtual cache. The virtual host, however is always allowed to read the virtual cache in order to satisfy the page request coming to it, i.e. if the requested page is not found in its cache (non virtual part),then it will also search the virtual cache (parts given to clients) to see if the page is there. If the page is found, then it reads the page and satisfies the request. A virtual host may give part of its cache to more than one client. Each client will have read/write permission to the virtual cache assigned to it. The virtual client host will have write permission only *to* non virtual part of its own cache. Once the virtual cache has been given to a client, only the client can read from or write to the virtual cache allocated to it.

A client node can acquire virtual cache from more than one virtual host. While receiving a request from client to retrieve data from virtual cache, the host behaves as a virtual origin server for that data object. However the above overhead associated with virtual caching is not only well compensated rather suppressed by reduction in average latency at the proxy servers because of better cache sharing.

### *2.1.2 The Protocol*

Following is the precise description of events and associated actions for the client and the host nodes of the virtual caching scheme.
CLIENT END EVENTS and ACTIONS

**Event 1 :**When a client node receives or generates a request for a page. The client searches its cache (local cache and virtual cache if any) for the page. Here the local cache means entire physical cache, and by virtual cache we mean the table containing list of pages stored in the virtual cache. We call this table the virtual cache page table.
Case 1: Page is found in the cache.
Case 1.1: Page is available in the local cache.
Action: Request is to be satisfied by reading the page from the cache in the conventional manner.
Case 1.2: Page is found in the virtual cache.
Action: Client sends a PAGE-RETRIEVE request with the PageID to the host node.

Case 2: Page is not found in the cache (neither in virtual nor in local cache). Action: Client sends a request for the page to the upper level node in the caching hierarchy towards the origin server.

**Event 2:**When a page is brought from the origin server. The client decides if the page is to cached based on the diffusion policy (such as D4, harvest or any other). If the page is to cached then clients first tries to cache the page in its local cache.

Case 1: There is enough empty space in the local cache for caching the page. Action: Cache *tpe* page in the local cache in conventional manner.

Case 2: The local cache is full
Action: Check if there is enough space available in the virtual cache. If the client has virtual cache at more than one hosts then this checking is done in the order of increasing cost in terms of time of accessing the virtual cache. In other words, preference is given *to* the virtual cache from where the cached page can be retrieved faster. If enough space is available in the virtual cache, then send the page to the host with a PAGE-INSERT request to insert the page in the virtual cache, The client marks this page as "sent to host for caching" and makes and entry about the page in the acknowledgement table. This acknowledgement table lists all those pages, which have been sent to some hosts for caching in the virtual cache, but the acknowledgements have not been

received from the host.

Case 3: Clients finds that both its local cache and virtual cache are full and there is no space in the cache to store the new page.

Action: Decide which page to replace (choose the victim page) considering pages in local as well as virtual caches all at a time. This decision is taken on the page replacement strategy (LRU, LFU, FIFO or other) followed,

Case 3.1: The victim page is in local cache.

Action: Replace the victim page with the new page in the conventional manner.

Case 3.2: The victim page is in virtual cache.

Action: Send a PAGE-REPLACE request to the host along with the new page to be cached and PageID of the victim page that is to be replaced. Remove the entry for the victim cache page table and mark the new page as "sent to host for caching", i.e. make an entry about the page in the acknowledgement table.

**Event 3 :**When an acknowledgement is received from some host.

Action: The client retrieves the pageID from the acknowledgement and removes the page entry for that page from the acknowledgement table and makes entry for the page in the virtual cache page table (this table lists the pages cached in its virtual caches).

**HOST END EVENTS and ACTIONS**

**Event 1** :When the host receives a PAGE-RETRIEVE request.

Action: The host checks if the client is a valid client. If yes, then it checks the page table for that, client. If page is found, then it reads the pages and sends back to the client. In case the client is not a valid client or the requested page is not found in the virtual cache of the client, an error message is generated and sent to the client.

**Event 2 :** When the host receives a PAGE-INSERT request.

Action: The host checks the validity of the client as done for the event-l done above. If the client is a valid client, then it checks if there is enough unused space in the virtual cache of the client to the cache page. If *yes,* then it writes the page to the cache and sends acknowledgement to the client that page has been cached. It makes an entry for this page in the" page table for the client.

**Event 3** :When the host receives a PAGE-REPLACE request.

Action: If the client is valid, then the host checks if the victim page is there in the virtual cache of the client. If yes, then the victim page is replaced by the new page received from the client and an acknowledgement is sent to the client for this action. However, if the victim page is not found in the virtual cache of the client or the client is not a valid client, then an error message is generated.

The client nodes can be considered as heavily loaded nodes looking for the host nodes can be thought of as lightly loaded nodes to give off their load. Looking the cache allocation from this perspective we did an in-depth study of existing load balancing algorithms.

## 2.3 Resource Discovery Algorithms

Resource discovery was first defined in [22],as the {ask to compute the connected components in the underlying graph of *Go* (where the underlying graph is the undirected graph obtained from *Go* by removing the direction from all arcs). More formally, The input to the problem is a directed graph Go(V,Eo). Each vertex (network node) knows (has a list of) all its outgoing arcs (but not its incoming arcs). A distributed algorithm is said to solve the Resource Discovery Problem if the following applies to every weakly connected component C in the directed graph G when the algorithm terminates:

(a) there exists a vertex (termed *root)* v in C such that for every other vertex u in C, G contains a directed arc (v, u) (or in other words, v knows all the ID's in C);

(b) every vertex u in C "designates" vertex v as the unique root of the component (in the implementation a variable called *PTR(u)* is set to the ID of v).

### *2.3.1Flooding Algorithm*

According to [22], this algorithm is widely used by internet routers and where every node acts as a transmitter and receiver and every node tries to send every message to every node of its neighbor, a newly added new edge is not used for any communication, direct communication exists only in between initially existing set of neighboring edges of the network. The required number of rounds of this algorithm is equivalent to the diameter of the graph. So *H archal et. al* claimed that this algorithm can be very slow if not started with a graph, which has small diameter.

### *2.3.2 The Swamping Algorithm*

According to [22], swamping algorithm is similar to flooding algorithm except this algorithm allows a node to connect with all of its current neighbors, not only with the set of initial neighbors. *Harchal et. at.* suggested that the main advantage of this algorithm is this algorithm needs *O(/og n)* rounds to conyerge to a complete graph and which is irrespective to the initial configuration. However the disadvantage is communication complexity of this algorithm grows very quickly.

### *2.3.3 The Random Pointer Jump Algorithm*

In this algorithm, in each round, each node contacts with a random neighbor, and then this random neighbor sends all of its neighbors to the sender node. Finally sender neighbor and random neighbor's neighbors get merged. [22] claimed that a strongly connected graph with *n* nodes needs *9(n)* complexity time to converge to a complete graph.

## 2.4 Load Distribution Approaches

Livny and Melman [13] showed that probability *P* that the system is in a state in which atleast one task is waiting for service and atleast _one server is idle is high, indicating good potential for performance improvement through load distribution. At high system utilizations, the value of *P* is low as most servers are likely to be idle, which indicates lower potential for load distribution. Similarly at low system utilizations, the value of *P* is low as most servers are likely to be idle, which indicates lower potential for load distribution. Load distribution seeks to improve the performance of the distributed system usually in terms of response time or resource availability by allocating workload amongst a set of co-operative hosts.

### *2.4.1 Load Balancing*

Load Balancing tries to ensure that every processor in the system does almost the same amount of work at any point of time. Processes might have to be migrated from one machine to another even in the middle to ensure equal workload. Algorithms for load balancing have to heavily rely on the assumption that the information available at each node is quite accurate, in order to prevent processes from being endlessly circulated about the system.

### *2.4.2 Load Sharing*

Load sharing scheme {2] is a weaker version of the load balancing which tries to initiate a process to lightly loaded node

and hence distribute the overall load of the system to its individual nodes using only non-preemptive transfer of processes. Although load sharing doesn't ensure equal workload for every node in the system, it is easier to implement and ,can more easily accommodate heterogeneity in the system.

### *2.4.3 Hierarchical Balancing Methods*

The Hierarchical Balancing Method organizes system into hierarchy of balancing domains, thereby decentralizing the balancing process. Specific processes are designated to control the balancing operations at different level of hierarchy. The hierarchical scheme distributes the load balancing responsibilities to all the processors in the system. It is effective for balancing local load imbalances as well as excessive global balances.

### *2.4.4 The Gradient Model*

The basic concept of this approach is that under-loaded processors inform other processors in the system about their state and over-loaded processors responds by sending a portion of their load to their nearest lightly loaded processors in the system. The resulting system is the form of relaxation where task migration through the system is guided by the proximity gradient and gravitates towards the under-loaded points in the system. The scheme is based on two threshold parameters: the Low Water Mark (LWM) and High Water Mark (HWM). A processor state is considered lightly loaded if its load is below LWM and heavily loaded if its load is above HWM, and moderate otherwise. A node proximity is defined as the shortest distance from itself to the nearest lightly loaded node in the system. On the basis of proximity the transfer decision are taken.

### *2.4.5 Nearest Neighbor Algorithm*

With nearest neighbor load balancing algorithms, a processor makes balancing decision based on localized workload information and manages workload within neighborhood. Nearest neighbor load balancing algorithms rely on successive approximation to global uniform distribution; hence each operation need only be concerned with direction of workload migration and the issue how to apportion excess workload. The diffusion and dimension exchange methods that fall in this category are discussed. With diffusion method, a heavily or lightly loaded processor balances workload with all of its neighbors simultaneously in load balancing operation. Cybenko [20] showed that diffusion method eventually coerce any initial workload distribution into global uniform distribution in static situation "in which no workload are generated or consumed during load balancing. Although the result of both theoretical and experimental study point to the superiority of dimension exchange methods in hypercubes, it might not be the case for other popular networks [1]. The most of the study is made about the synchronous implementation of these algorithms. Local average diffusion and optimally tuned diffusion are the modified diffusion methods. Dimension Exchange methods outperform diffusion methods in synchronous implementations.

**Dimension Exchange Algorithm:**

With the dimension exchange method, a processor in need of load balancing balances its workload with its neighbors one at a time a new workload index is computed, which will be used in subsequent pair wise balancing [10].

With the dimension-exchange method, any processor which invokes a load balancing operation balances its workload with its neighbors successively. For a processor i, it works in the following way.

*f=for( c=1;c~d( i);c++)* $W_i = W_i + \lambda(W_{jC} - W_i)$

where $j c \varepsilon A(i)$; and $0<\lambda<1$, called the dimension-exchange parameter, is given a fixed value beforehand which determines the fraction of excess workload to be migrated between a pair of processors. The formula says that a balancing operation in the dimension-exchange method comprises d(i) pair wise balancing steps for processor i where *d(i)* is the degree of node i. At each step, processor i balances its workload with one of its neighbors, and uses the new result for the subsequent balancing. It is because of the sequential nature in the sequence of balancing steps, a load balancing operation requires *d(i)* communication steps in both the all-port and the one-port communication models. The efficiency of the dimension-exchange method is determined by the dimension exchange parameter. A dimension-exchange operation with different choices of the parameter will reduce the workload variance of the system by different degrees. Two choices of the parameter have been suggested as rational choices in the literature,

**a) Average dimension exchange (ADE)**

**b) Optimally tuned dimension exchange (ODE)**

The dimension exchange method can be implemented without difficulty in cases where only a few processors that are not close to each other are in need of load balancing at the same time .However its synchronous implementation requires processors to be coordinated in order to parallelize balancing operations along different communication channels as well as to avoid communication collisions. The parallelization of pair wise balancing operations can be realized by partitioning the set of edges into a number of subsets such that no two adjoining edges are in the same subset. The pair wise balancing steps along the channels in the same subset can then be performed concurrently without collisions. Such graph partition is equivalent to the problem of edge coloring of graphs.

**Diffusion Exchanged Algorithm**

With the diffusion method, a heavily or lightly loaded processor balances its workload
with all of its nearest neighbors simultaneously in a load balancing operation [1].

With the diffusion method, any processor which invokes a load balancing operation compares its workload with those of its nearest neighbors, and then gives away or takes in certain amount of workload with respect to each of nearest neighbors.

The diffusion operator in a processor i can be written in the form

$$Fi(.) == Wi + \Sigma \alpha_{ij}(W_j - W_i)$$
$$l \varepsilon A(i)$$

where $0 < \alpha_{ij} < 1$,called the diffusion parameter, is predefined to dictate the portion to be migrated between any two processors. Processor i apportion excess workload

*I* $W_j - W_i$ *l* to processor *j* if $W_j > W_i$, or fetches some workload from processor *j* otherwise. Clearly, a load balancing operation with the diffusion method requires only one communication step in the all-port communication model, but *d(i)* steps in the one-port communication model. As in the dimension-exchange method, the efficiency of the diffusion method is determined by the diffusion parameter. Following are two common choices of the parameter.

a) Local average diffusion(ADF) b)optimally tuned diffusion (ODF)

## Chapter 3

## Resource Discovery and Allocation

## algorithms for Bounded Hops

### *3.1 Problem Statement*

In the bounded hops version, of the problem there is a bound on the hops for the resource deficient nodes to look for resource-surplus nodes in the network.. Resource deficient nodes can look for resources with a finite number of hops only. Clearly, such a solution obtained doesn't give an optimal result.

*Problem P* 1: - Resource discovery and allocation problem for finite hops.

There are n nodes in a connected undirected network *G=( V,E).* Assume that each edge *e =(i,j)* has an associated non negative real valued weight, *weight(e)=weight*$_{ij.}$ We assume that for all i and *j*, *weight*$_{ij.}$*= weigh*$_{iij.}$. Here weights represent the cost of communication between the nodes. Each node *1li* is having some capacity (resource owned by the node) Ci and some requirement (resource required by the node) *Ti'* Capacity of each node may be equal or less or greater than the requirement. For sake of simplicity assume that sum of all capacities and requirements over all nodes across the network are greater than or equal; implying that total requirement can be met within the network. Let *R* be set of all the nodes whose requirement of resource is more than their own capacity i.e. *R*= {ni : $r_i > c_i$;}. Let *S* be the set of all the nodes whose capacity is more than their requirement i.e. *S={* ni : $r_i < c_i$ }. Let *T = [*$t_{ij}$*]* (where *i=1..n* and *j=1..n)* be the transfer matrix denoting the amount of resource which is transferred. $T_{ij} < 0$ means node ni will receive $T_{ij}$, units of resources from node nj. $t_{ij} > 0$ means node ni will give $T_{ij}$ units of resources to node $n_{j.}$

Task is to devise an efficient algorithm (whose communication complexity is less than $O(N^2)$ ) to minimize the sum, $\Sigma$ $((r_i - c_i) + \Sigma\ t_{ij})$

$ni \varepsilon R$ $\quad$ nj$\varepsilon$S

under the constraint that $\forall\ ni\ \varepsilon\ S, ((\ c_i - r_i) - \Sigma\ t_{ij}\ ) >= 0$ i.e.

$\forall\ nj \varepsilon R$

we have to satisfy requirement of resource deficient nodes under the constraint that nodes having surplus resources share resources among resources deficient nodes in such a way that they themselves don't become resource deficient. Also resource deficient nodes do not accept more resources than their requirement i.e.

$\forall\ ni\ \varepsilon\ R, (\ r_i - c_i) - \Sigma\ t_{ij}\ ) >= 0.$

$\forall\ ni\ \varepsilon\ S$

In order to compare the optimality of our resource distribution algorithm we are using the solution given by Integer Linear Programming (ILP) as benchmark. Solution given by (ILP) is the best solution which is possible under the restriction of bounded hops. In this study we are solving the problem for 1 hop.

### *3.2 Model of Computation*

We consider message passing systems with no failures. In a message passing system, processors communicate by sending messages over communication channels, where each channel provides a bidirectional connection between two specific processors. furthermore, we assume our timing model to be synchronous, i.e. processes in the system run in lock step manner, where in each step, a process receives messages (sent to it in the previous step), performs a computation, and sends messages to other processes (received in the next step). In synchronous computation, a process knows all the messages it expects to receive messages and perform computation at any time.

### *3.3 In case of Non-Anonymous Arbitrary Topology*

Resources are distributed in the overlapping balancing domain by allowing them to proceed in circular wait condition. We had also seen that this approach may cause deadlock We can also avoid deadlocks by preventing circular wait condition from occurring. One way to ensure circular wait condition never holds is to impose a total ordering of all I*_AM_STARTING* requests sent by donor nodes and to require that each deficient node Sends an *OK* message in increasing order of enumeration of the request. This can be easily done by assuming that each node in the network has a unique ID (which can be thought of as an IP address) and by associating this ID with each message sent by the node. So when a deficient node receives *L_AM_STARTING* messages from more than one donor node, a deficient nodes sends an *OK* message to the donor nodes in their increasing order of IDs of the respective sender nodes. The overlapping balancing domains proceed one after the other. After the distribution occur in each balancing domain, it sends an *LAM_DONE* message. When the root of the spanning tree receives *LAM_DONE* message from all the balancing domains, it knows that the resource distribution has finished in the balancing domains and consequently sends *Terminate* message.

## Chapter 4

### 4.1 Proposed Algorithm for Anonymous Arbitrary Topology

In this algorithm , each node at some point of the time holds a token called DELEGATKLEADERSH1P _TOKEN. The node which has the token is the leader and has the privilege to delegate leadership by passing the token. The leader can distribute resources if all of its children have been visited. When the token visits each node takes some decision depending upon the-fact that the receiver node is a donor node or deficient node. If the receiver node is a donor node, it distributes its surplus resources among its immediate deficient neighbors. Otherwise if the receiver node is a deficient node, it just forwards the token. The token traverses the node of the spanning tree in a post order so that each node receives the token at most twice. The idea behind using the post order traversal is that we want a node to distribute resources only if all of its child nodes have distributed resources. .

The proposed algorithm for anonymous arbitrary topology works in two phases.

**1. Spanning Tree Construction:** A distributed spanning tree construction algorithm converts the arbitrary topology into a tree topology. In the spanning tree, each node is aware of its parent and of its immediate child nodes.

**2. Resource Distribution Phase*:*** Initially root node *r* of the tree is having the DELEGATKLEADERSHIP _TOKEN. Root node then sends the token to one of the unvisited child node i. Upon receiving the token, the unvisited node i marks itself visited. If the node i has some unvisited child node *j* it forwards the token to it. Otherwise if the node i has no unvisited child node it

distributes the resources among its immediate neighbors.

Resource distribution is done as follows. The donor node i first sends LAM_STARTING message to all of its resource deficient neighbors. Each deficient neighbor node upon receiving the message sends OK message along with its amount of resources requirement. Upon receiving OK messages from all the deficient neighbors, the donor node serves request according to the priority of their respective sender node. Priority of the sender node is inversely proportional to the number of its donor neighbors. The donor node the sends GIVE messages along with the amount of resources given to all its immediate deficient neighbors to distribute the resources. After resource distribution, the donor node sends DELEGATE-LEADERSHIP _TOKEN message to its parent, if no parents exists the root node sends TERMINATE message to all of its children to indicate the termination of the algorithm.

### *4.1.1 Pseudocode of the Proposed Algorithm for the Resource Distribution Phase Anonymous Arbitrary Topology*

Resource distribution phase starts when the node receives TERMINATE_PHASELMSG of spanning tree construction phase. TERMINATE-PHASELMSG marks the end of spanning tree construction phase and the beginning of resource distribution phase.

**Initial state at all nodes at the beginning of resource distribution phase**

{
Each node knows which of its neighbors are its parent and children in the spanning tree. Each node also knows which nodes are its immediate neighbors and. whether they are donor nodes or Deficient nodes. Root node is having the DELEGATE_LEADERSHIP _TOKEN.
}
When a node has a DELEGATE-LEADERSHIP _TOKEN
{
If all of the children of receiver node have been visited {
If the receiver node is donor node. {
If resource deficient node exist. {
Send LAM_STARTING_MSG to all the currently deficient nodes. Note that it is possible that node which were deficient earlier are no longer deficient, as they could have got resources from some other donor nodes.
}
Else if no resource deficient neighbors exist {
Send to TERMINATE message to all the children,
if the receiver node is ROOT node.
Otherwise send DELEGATE_LEADERSHIP _TOKEN to the parent node.
}
}
Else if the receiver node is a deficient node
{
If the receiver node is ROOT node
{ Send TERMINATE message to all the children.

}
If the receiver node is not the ROOT node {
Mark the receiver node has been visited.
Send DELEGATE_LEADERSHIP _TOKEN message to the parent.
} }
}
Else if the some children of the receiver node are unvisited
{
Send DEl.EGATE_LEADERSHIP -TOKEN message to the unvisited child node.
}
}
When a node receives I_AM_STARTING_MSG
{
Receiving node marks LAM_STARTING~ISG has been received. Mark that the reply OK message has been sent and send OK message to sender node.
}
When a node receives OK_MSG
{
Record the number of neighboring donor nodes, sender node has.
Mark OK_MSG has been received from the sender node.
If Receiver node has received all OICMSG from neighboring resource (as of now) deficient nodes
{
Sort deficient neighboring nodes in ascending order of number of donor nodes they have.
(This is the decreasing order of the priority of resource deficient nodes.)
For each of the neighboring deficient node, if their OK_MSG contains some request for resources($\geq 0$)
{ send GIVE_MSG with resources given in decreasing order of priority. }
If receiving node is a ROOT node.
{ Send TERMINATE message to all of its children.

}
Else if receiving node is not a ROOT node.
{ Send DELEGATE_LEADERSHIP _TOKEN message to the parent.
} }

Else { Wait for all OK_MSGs to arrive.
}
}
**When a node** receives GIVE_MSG
{
Receiver node marks GIVE_MSG has been received from the sender node. Receiver node records the amount of resources received from the sender node.
}
**When a node** receives TERMINATE_MSG
{
Send TERMINATE_MSG to all its children.
}

### *4.1.2 Complexity Analysis of the Resource Distribution Phase of the Proposed 1- Hop Algorithm for Anonymous Topology*

**Message Complexity:** Resource distribution in each balancing domain requires passing of constant number of messages on each edge between donor node and deficient node. To be more precise during resource distribution at most 5 messages pass over each edge (1 *LAM_STARTING* message, 1 OK message and 1 *GIVE* message, 1 *LAM_DONE* message, 1 *TERMINATE* message). If $N_d$ is the number of donor nodes and K is the maximum degree of a donor node (in the tree topology). Then message complexity of the resource distribution phase of the algorithm is $O(N_d K)$ which in turn is O(E) as all the resources are distributed along the edges of the network.

**Bit Complexity:** Each message has 0 (log *N)* bits for representing the node to which the message is meant. Each node also contains O(SIZE) bits for representing the amount of requirement and capacity of resources where SIZE is a constant. So size of each message is O(log $N$ + SIZE).

Therefore bit complexity of the resource distribution phase is O(N(log $N$ +SIZE)) i.e. O(N(log *N))* bits.

**Time Complexity:** Load distribution in singular domain takes 0(1) time. Overlapping domains in which resource-surplus nodes are adjacent also share resources as singular domains and hence take 0(1) time. However overlapping domains in which resource-surplus nodes are non adjacent, proceeds one after the other synchronously. Resource distribution takes $O(K_o)$ time where $K_O$ is the maximum number of overlapping domains. Time taken by token DELEGATE-LEADERSHIP _TOKEN return to the root node after visiting all the nodes of the network is O(N). Thus overall time complexity of resource distribution phase for anonymous network is $O(K_O + N)$ which is O(N).

## Chapter 5

### *Conclusion :*

In this paper to address the virtual cache allocation problem, we formulated a general resource discovery and allocation problem. This formulation is general in the sense that we haven't made any assumption specific to cache distribution and hence the proposed heuristics can be used to distribute any static resources. we presented the basic scheme and protocol of virtual caching scheme. Different recent resource discovery algorithms, load distribution approaches were presented. Clearly none of the load distribution and resource discovery approaches could be applied to the problem we have formulated to minimize the amount of unfulfilled request of deficient nodes. By using non-anonymous arbitrary topology with sequence number of request to resolve deadlocks and distributing resources over the original arbitrary network. Sequence number of the request is the unique ID of sender node. The basic assumption that each node in the network has a unique ID in the non anonymous algorithms has been relaxed here. So, each node in the network may not necessarily have unique ID. We proposed a heuristic to distribute resources over anonymous arbitrary topology by passing a token .The token is privilege to distribute the resources. We gave a complete complexity analysis of the proposed algorithms.

## Chapter 6

### *References:*

[1] Xu, B. Monien, R. Luling, F. C. M. Lau : Nearest Neighbour Algorithms for Load Balancing in Parallel Computers *Concurrency: Practice and Experience, Vol.* 7, *No.7, pp. 707-736, Oct.* 1995..

[2] Yung Wang and Robert Morris: Load Sharing in Distributed Systems. *IEEE Trans. on Computers, pp 204-217, March* 1985.

[3] Jorge Escorcia, Dipak Ghosal, Dilip Sarkar: A novel cache distribution heuristic algorithm for a mesh of caches and its performance evaluation. *Computer Communications* 25(3): *329-340 (2002)*

{4] Joydeep Chandra: Analytic and Simulation studies on Effect of Distribution of Caches in Networks.M. *Tech. Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, 721302, India, Jan. 2002.*

[5] Mohapatra, Pradosh. : Fully Sequential and distributed dynamic algorithms for Minimum Spanning Tree. *Computing Research Repository, Feb .2000.*

[6] Santoro, N. : On the Message Complexity of Distributed Problems *Int. Journal of Compo and In! Sci.* 13. 1984, *pages* 131-147

[7] Casavant, T.L., and Kuhl J.G.: A Taxonomy of Scheduling in General Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering, VOL* 14, *No* 2, *February* 1988

[8] Bubendorfer, K.P : Resource based policies for load distribution *Ph.D Thesis, Vic toria University of Wellington, August* 1996..

[9 Balter, Harchol and, T. Leighton, and D. Lewin: Resource Discovery in Distributed Networks. *In Proc. 15th ACM Symposium on Principles of Distributed Computing,Technical May* 1999 *pp229-237*

[10 RG.Gallager, P.A. Humblet, and P.M. Spira: A distributed algorithm for minimum weight spanning trees *A CM transactions on Programming Language and Systems,PP*66-77.