Journal of Theoretical and Applied Information Technology
15th May 2026. Vol.104. No.9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TREE TOPOLOGY-BASED RESOURCE ALLOCATION AND DISCOVERY ALGORITHM FOR ONE HOP

[1]MUSA DIMA GENEMO, [2]DR. YALAMANCHILI SAROJA, [3]KISTAM GOPI, [4]VEERAMANI MUTYAM, [5]DR BHARGAVI PEDDI REDDY, [6]DR.RAMIREDDY NAVATEJAREDDY, [7]SRINIVASA RAO MADALA, [8]NIMMAGADDA YASWANTH SAI, [9]K. KOTESWARARAO

[1]Associate Professor, Department of Software engineering, Gumushane University, Turkey
[2]Assoc professor, Dept. of Mathematics, Malla Reddy Technical Campus, Malla Reddy Vishwavidyapeeth (Deemed to be University), Hyderabad, Telangana, India
[3]Assistant professor, School of Computing, Amrita Vishwa Vidyapeetham, Amaravati, A.P, India
[4]Assistant Professor, Department of CSE, Aditya University, Surampalem, Andhra Pradesh, India
[5]Associate professor, Department of CSE, Vasavi college of engineering, Hyderabad, Telangana, India
[6]Professor, Dept of CSE, Saveetha school of engineering, Saveetha Institute of Medical and Technical Sciences, Tamilnadu, India,
[7]Professor, Dept. of CSE, Sri Harshini College of Engineering and Technology for Women, Ongole, AP,
[8]Asst.Professor, Dept of CSE, Koneru Lakshmaiah Education Foundation Vaddeswaram, AP, India
[9]Assioc Professor, Department of CSE, PVP Siddhartha Institute of Technology, Vijayawada, Andhra Pradesh, India

Email: drpriyakotesh@gmail.com

ABSTRACT

The components of a distributed system include multiple computing nodes which may use different types of hardware and connect to each other through a communication network that operates without shared memory or clock systems. The primary advantage of distributed systems enables organizations to share resources which results in faster computing times together with increased data availability and system reliability. The process of resource sharing requires organizations to first identify their resources and then assign those resources before anyone can access them. Computer systems implement caching to store frequently used data and program segments. The new caching technique called virtual caching enables a host node to delegate its cache authority for specific pages to nearby nodes who can use its partial cache space. The virtual caching protocol specifies neither the client node's process for accessing virtual cache nor its method for retrieving virtual cache from distant servers. The resource discovery and allocation problem needs us to develop a solution for this particular issue. The resource donation process requires us to find donor nodes who have extra resources while we need to assess which resources will be given to deficient nodes at each connected network point. Our goal is to achieve complete fulfillment of requests which deficient nodes make for resources. The importance of virtual caching lies in the fact that it doesn't require physical movements of cache. The requirements of the system determine when virtual cache allocation can be modified. The proposed heuristics provide efficient performance because they require minimal time and perform limited communication operations.

We assess distribution quality through a comparison between heuristic distribution results and the distribution outcomes from the ILP problem solution. We present and assess several heuristics which aim to reduce unfulfilled resource requests that arise from resource-deficient nodes during their resource search process which has a maximum distance of two hops. The paper restricts its analysis to single hop operations only. The paper demonstrates through its research that this algorithm reaches optimal solution status based on our established benchmark criteria. The resource distribution for bounded hops restricts its operations to one-hop resource distribution. Every resource-surplus node transfers additional nodes to other nodes while maintaining its own resource balance without reaching resource-deficient status. Load distribution cannot be divided into infinite parts. Our research focuses on measuring the fulfillment level for each resource-deficient node request.

Key words: *Donor, Deficient, Caching, Optimal, Topology, Resource Allocation*

1. INTRODUCTION

A distributed system consists of multiple processors which operate independently and connect through a communication network [1]. The distribution system provides its primary benefit because users can share resources with other users. The team needs to identify all resources because they need to implement proper allocation methods for resource utilization and resource distribution. The process of resource utilization becomes more effective through the implementation of caching. Computer systems implement caching to store frequently accessed data in specific memory locations. Virtual memory systems function as secondary storage systems that cache data to main memory systems with higher speed. The virtual caching system was introduced in distributed environments as a direct application of this fundamental principle. Virtual caching enables a host node to share its caching authority for specific pages through a designated portion of its cache with nearby network nodes. Virtual caching operates similarly to virtual memory because both systems provide users with enhanced access to virtual address spaces which enable them to map locations that exist beyond their physical address limits [2].

The Virtual Caching scheme represents a new approach to caching. The virtual caching scheme designates a caching node as its virtual host which provides access to its cache resources. The virtual host needs to give up some of its cache capacity so that virtual clients can access the shared space. Virtual hosts donate their extra resources to operate as donor nodes. Virtual clients can be considered as deficient nodes which need resources. The virtual caching protocol does not define the process through which a client node retrieves virtual cache from the remote host. We establish our solution through the introduction of multiple heuristics which we evaluate for their effectiveness [3].

1.1 Resource Discovery and Allocation Problem

The virtual client nodes function as deficient nodes because they need to search for resources through their cache access. The virtual host nodes function as donor nodes because they provide resources through their cache availability. We define resource discovery and allocation as our problem formulation under this paradigm [4].

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 communication 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,junits 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,

$$\Sigma((ri - Ci) + \Sigma (t_{ij}) \text{ under the constraint that}$$
$$ni\varepsilon R \qquad nj\varepsilon S$$
$$\forall\ ni\ E\ S,\ \langle Ci - ri) - \Sigma\ t_{ij}) >= 0 \text{ i.e.}$$

we have to satisfy $\forall\ nj\varepsilon R$

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 cannot 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 Σ (ri - ci) >=$\Sigma(r_j - C_j)$ *is 0* otherwise the value of $ni\varepsilon S \qquad nj\varepsilon R$

metrics is Σ (ri - ci) -$\Sigma(C_J - R_J)$.
$ni\varepsilon S \qquad nj\varepsilon R$

We use benchmark metrics to assess our algorithms because they provide us with a method to evaluate distribution quality. The restriction requires donor nodes to distribute resources in a manner that prevents them from reaching resource deficiencies. The deficient nodes have a limit on resource intake which matches their actual needs. We aim to decrease unfulfilled requests from deficient nodes while maintaining efficient message processing and time management. The resource discovery problem has been transformed into an optimization problem which we will solve using an ILP solver. The solution obtained through this process will be used to compare the results of heuristics based on their unfulfilled request distribution.

The Internet of Things (IoT) infrastructure and wireless sensor networks (WSN) and ad hoc networks require efficient resource allocation and discovery systems to deliver optimal performance results. The systems depend on decentralized self-organizing methods because centralized control fails when there are latency issues and system overhead problems and needs to achieve fault tolerance. The resource allocation and discovery algorithms development requires teams to face difficulties because of the network's design which uses limited identifiers and allows anonymous nodes.

The standard network topology requires multiple nodes to share resources which they must discover and use while avoiding excessive system overhead and operational conflicts. The one-hop communication model limits each node's ability to communicate with its immediate neighboring nodes. The system achieves communication simplicity through range and latency reductions but visibility restrictions create difficulties for coordination especially when node identities remain concealed or hidden. Tree-based topologies provide a beneficial solution to this problem.

A tree topology creates a hierarchical system for organizing nodes which enables users to traverse the network efficiently and aggregate data and broadcast information. The operational advantages of tree structures which exist in multi-hop routing can be used in one-hop environments through the implementation of logical tree formation instead of physical multi-hop distance. The virtual tree representation enables nodes to hold a simple hierarchical structure which improves their ability to discover resources and distribute resources compared to non-hierarchical flat topologies.

The Tree Topology-Based Resource Allocation and Discovery Algorithm (TTRADA) which this research presents uses tree structures to control resource distribution through tree structures. The network demonstrates improved coordination through tree structure implementation which enables better control through defined paths and enables better handling of resource conflicts. The system assigns nodes their roles as parent or child or leaf based on the logical criteria of resource type and resource availability and task priority.

TTRADA operates in three main phases: (1) Tree Initialization, where nodes organize themselves into a logical tree based on pre-agreed criteria or randomized roles; (2) Resource Advertisement and Discovery, where nodes propagate resource availability and requests up and down the tree hierarchy; and (3) Allocation Decision, where parent nodes arbitrate resource conflicts among their children to ensure fair and efficient access.

The method gives three primary advantages because it establishes special communication paths through tree branches which decrease unnecessary network traffic together with its decision distribution capability which enables system expansion and its parent node work management system which creates equitable system operation. The structure supports dynamic environments through its ability to recalculate system operation after each node failure and through its regular recalibration process.

The implementation of tree-based algorithms within anonymous networks encounters multiple difficulties which include preventing loops while assigning roles to users who lack identifiers and preserving network structure during user turnover. The research presents a solution to the existing problems through its three main components which include probabilistic role negotiation and localized decision protocols and the lightweight tree maintenance techniques which work within one-hop limit. The research presents a new approach for organized resource distribution and exploration in one-hop anonymous environments which uses tree-topology methods. The proposed algorithm aims to strike a balance between order and decentralization, offering a scalable, fair, and efficient solution for resource management in modern decentralized systems.

2. VIRTUAL CACHING SCHEME

2.1. Schematic of Virtual Caching

Virtual caching allows users to access only a specific portion of cache memory from nearby nodes which are not currently using their complete cache capacity. Virtual hosts must give up control

over their cache space because other nodes will use this shared cache space to store their data. The virtual cache of node A enables it to use cache memory from another network node B. 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. The virtual cache of node A at B represents the cache space that A has reserved from B. Nodes A and B are called client and host, respectively. The client can view a larger cache than what its actual physical cache provides. A client starts to access a page by searching its physical cache and then continues to search its virtual caches which it has borrowed from other host nodes.

The client who receives cache space from the virtual host will control all rights for writing data to the virtual cache. The virtual host requires access to the virtual cache because it needs to fulfill page requests. The system will first check the non-virtual cache for page retrieval. The system will then check the virtual cache which contains client-accessible elements. The system will read the page content when it detects a page. A virtual host may give part of its cache to more than one client. The virtual cache assigned to each client will allow them to access both reading and writing functionalities. 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. The host functions as a virtual origin server for that data object when it receives a request from the client to retrieve data from the virtual cache. The system achieves better cache sharing which results in shorter average latency at the proxy servers through this overhead reduction.

2.2 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 : The client starts its page request process when it either receives a request from another node or creates a request itself. The client searches its cache (local cache and virtual cache if any) for the page. The local cache refers to the complete physical cache while virtual cache refers to the table that holds information about stored pages 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: Virtual cache space needs assessment to determine available storage capacity. The system checks virtual cache access at multiple hosts according to ascending time costs needed to reach virtual cache. The system selects the virtual cache that enables faster cached page retrieval from its storage. The system sends the page to the host through a PAGE-INSERT request to insert the page into the virtual cache when sufficient virtual cache space exists. The client designates this page as "sent to host for caching" while entering the page information into the acknowledgement table. The acknowledgement table displays all pages which have been sent to virtual cache locations at various hosts for caching yet their host acknowledgements remain unreceived.

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 verifies whether the client meets the requirements of a valid client. The host proceeds to check the page table records for that particular client after confirming the client is valid. The system reads the pages when it finds the page and then delivers them back to the client. The system generates an error message which it sends to the client when either the client fails validation or the requested page cannot be located in the client's virtual cache.

Event 2 : The host receives a PAGE-INSERT request at which time it begins its operational process. The host needs to check client identity through the same procedure which was used to validate the event-l client. The system checks valid client status through two tests first testing system validity then determining whether virtual cache space can accommodate cache page requirements. The system transfers the page to cache memory while sending a confirmation message to the client about successful page caching. The system creates a page entry for the client in the page table.

Event 3: When the host receives a PAGE-REPLACE request.
Action: The host system checks for valid client status before proceeding to check whether the virtual cache of the client contains the victim page. The system updates the virtual cache by replacing the victim page with the new page which the client sent while sending an acknowledgment to the client about this update. The system generates an error message when the client fails to meet valid client requirements or when the virtual cache does not contain the victim page.

The client nodes function as heavily loaded nodes while the host nodes operate as lightly loaded nodes that distribute their processing requirements. Our research team conducted a comprehensive analysis of existing load balancing algorithms to examine cache allocation from this specific angle.

2.3 Resource Discovery Algorithms

Resource discovery was first defined in ,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). The problem input consists of a directed graph Go which has vertex set V and edge set 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 fol¬lowing 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).

Flooding Algorithm

According to [5], this algorithm is widely used by internet routers and all nodes operate as both transmitters and receivers while every node attempts to send their messages to all neighboring nodes and the newly added edge remains inactive for communication purposes because direct communication functions only between the network's preexisting neighboring edges. The algorithm requires a number of rounds which matches the graph's diameter. So H archal et. al claimed that this algorithm can be very slow if not started with a graph, which has small diameter.

The Swamping Algorithm

According to , 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 converge to a complete graph and which is irrespective to the initial configuration. The algorithm has an advantage that it uses O(log n) rounds to reach a complete graph but this advantage is countered by its rapidly increasing communication complexity..

The Random Pointer Jump Algorithm

The algorithm requires each node to establish contact with a random neighbor during each round of its operation. The random neighbor transmits its complete list of neighbors to the sending node. The sender node combines its own neighbors with the neighbors of the random node. claimed that a strongly connected graph with n nodes needs 9(n) complexity time to converge to a complete graph [6].

Load Distribution Approaches

The research of Livny and Melman demonstrated that when the system reaches a state with pending tasks and at least one server remains unoccupied, the P value shows a strong probability, which indicates that load distribution would result in significant performance enhancements. The probability P value decreases at high system utilizations because most servers will remain unoccupied, which results in diminished capacity for load distribution. The probability P value decreases at low system utilizations because most servers will remain unoccupied, which results in diminished capacity for load distribution. Load distribution improves distributed system performance by distributing tasks among multiple cooperative hosts, which results in better response times and higher resource availability [7].

Load Balancing

Load Balancing attempts to distribute work equally among all system processors during every moment of operation. The system needs to transfer processes between machines at any time to achieve balanced workload distribution. The load balancing algorithms require accurate node information because inaccurate data will result in processes circulating throughout the system.

Load Sharing

The load sharing scheme in [8] functions as a less effective method of executing load balancing which establishes a process on a low usage node to share system resources among its nodes through non-preemptive process movement. Load sharing fails to distribute identical work among all system nodes but offers simpler implementation and better support for system diversity [9].

Hierarchical Balancing Methods

The Hierarchical Balancing Method organizes system into hierarchy of balancing do¬mains which enables the distribution of balancing activities across multiple regions. The system establishes dedicated processes which manage balancing tasks throughout its various hierarchical levels. The hierarchical scheme distributes load balancing duties to every processor within the system. The system effectively balances both local load discrepancies and major global balance issues [10]

The Gradient Model

The main idea behind this method states that processors which have low workload status will provide their current status information to all other processors in the network while the overloaded processors will dispatch part of their tasks to the nearest processors who have low workload status [11]. The achieved system functions as a relaxation system which directs task movements through the network by following proximity gradients and directs tasks to reach areas of lower system load. The scheme establishes two threshold parameters to operate which include Low Water Mark (LWM) and High Water Mark (HWM) as its fundamental components. 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. The team makes transfer decisions based on system proximity measurements [12].

Nearest Neighbor Algorithm

The algorithm requires a processor to use local workload data for making load distribution decisions while it handles operations within its immediate area. The operation of nearest neighbor load balancing algorithms depends on successive approximation which needs to achieve a global uniform distribution while each step requires only to determine the path of workload movement and the method for dividing excessive workloads. This section presents a discussion about diffusion methods and dimension exchange techniques. The diffusion method enables a processor with either high or low workload to execute load balancing through simultaneous interaction with all neighboring processors. According to Cybenko [13] diffusion method will transform any initial distribution of workload into a uniform distribution across all areas of a static environment "in which no workload are generated or consumed during load balancing. Theoretical and experimental results demonstrate that dimension exchange methods outperform other methods in hypercube networks although this advantage does not apply to common network types [14]. The most of the study is made about the synchronous implementation of these algorithms. The two modified diffusion methods use local average diffusion and optimally tuned diffusion as their base. Dimension exchange methods provide better performance than diffusion methods when used in synchronous operations.

Dimension Exchange Algorithm:

The dimension exchange method enables a processor to achieve load balancing through incremental workload distribution when it compares its current workload with each neighboring system. A new workload index is established for upcoming pairwise balancing operations after the current workload distribution has been completed [15]. The dimension-exchange method enables any processor which performs load balancing to distribute its workload to each of its

neighboring processors in successive steps. The system operates according to the following method for processor i. *f=for(c=1;c~d(i);c++) Wi = Wi + ɹ(WjC - Wi)*

The relationship requires that jcεA(i) while the dimension-exchange parameter which exists between 0 and 1 must receive its predetermined value which will define the proportion of excess workload that needs to be transferred between two different processors. The formula states that processor i needs to perform d(i) balancing operations which will achieve workload distribution across its processing units through the dimension-exchange method. Processor i uses its workload information to establish a workload balance with its neighboring processor at every step of the process [16]. The process of executing load balancing operations needs to handle d(i) communication steps under both all-port and one-port communication models because balancing steps follow a specific order. The dimension-exchange method functions according to its efficiency which depends on 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. The literature presents two parameter options which researchers consider as logical parameter selections [17].,

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 [18]. Higher communication collisions will occur during synchronous system execution because all processors must work together to balance their processing duties across various operational channels. The parallelization of pair wise balancing operations requires edge sets to be divided into multiple subsets that prevent two connected edges from existing within one subset. The pair wise balancing steps along the channels in the same subset can then be performed concurrently without collisions. The graph partition requires an edge coloring solution which functions as an equivalent mapping of graph edges to colors used for vertex connection determination [19].

Diffusion Exchanged Algorithm

The diffusion method enables a processor to distribute its processing tasks among its adjacent processors in order to achieve load balancing between its heavy and light processing states. The diffusion method enables a processor to perform load balancing by comparing its current workload to that of nearby processors which it considers as its nearest neighbors and then proceeding to distribute or receive workload from those specific processors [20].

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 [21].

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

2.2 Research Gaps: The tree-based topology algorithms provide structured and efficient resource allocation methods which enable resource discovery yet their application remains untested in one-hop anonymous networks. The Tree Topology-Based Resource Allocation and Discovery Algorithm (TTRADA) establishes a logical tree structure for single-hop environments but its practical implementation and performance enhancement require research to resolve multiple essential gaps. The first problem with anonymous environments involves their fundamental difficulty to create tree structures. The tree structure cannot be established because the nodes lack unique identifiers which makes it difficult to create stable parent-child relationships. The existing methods of tree structure creation do not work because they require partial identity knowledge or they depend on random role assignment which leads to suboptimal tree design. The development of lightweight identity-free tree construction methods requires a solution that preserves operational efficiency while preventing cycle and bottleneck issues.

The research area needs more examination of how changing network topologies will affect system performance when nodes move or fail or resources become unavailable. The dynamic behavior of nodes in one-hop networks makes it difficult to

maintain a tree structure that can adapt to changing conditions. Researchers have conducted only a limited investigation into dynamic one-hop environments which require tree maintenance and restructuring algorithms that should not create high operational costs.

The current algorithms found in existing systems develop tree structures without implementing fairness mechanisms to distribute load and allocate resources according to user priorities. The absence of proper resource arbitration leads to parent nodes gaining excessive access to resources while they systemically allocate resources to all connected child nodes. Future research should focus on studying how tree-based systems can implement decision-making processes which achieve fairness and distribute power among all members involved in the decision-making process.

The majority of tree-based research studies have been conducted using simulation environments which leads to a situation where only a few studies validate their findings through real-world testing. The existing research lacks experimental studies and benchmark assessments which need to evaluate TTRADA-like algorithms under actual conditions including packet loss and interference and energy consumption.

The process of closing these research gaps will create a foundation for developing tree-based resource discovery systems which can function effectively in anonymous decentralized environments.

3. RESOURCE DISCOVERY AND ALLOCATION ALGORITHMS FOR BOUNDED HOPS

3.1Introduction

The resource discovery and allocation algorithm for bounded hops supports resource discovery. The chapter presents three heuristics with their complexity analysis and the explanation of their mathematical foundations [22].

3.2Problem Statement

The version of the problem which includes bounded hops restricts resource deficient nodes to search for resource-surplus nodes within a specific distance limit. Resource deficient nodes can look for resources with a finite number of hops only. The obtained solution from this method does not produce optimal results.

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, $\Sigma ((r_i - c_i) + \Sigma\ t_{ij})$

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

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.

3.3 Model of Computation

We investigate message passing systems that operate without any operational failures. The message passing system enables processors to send messages through its communication channels which establish two-way connections between designated processor pairs. Our timing model operates under synchronous conditions which

require system processes to function in synchronized steps while they process incoming messages and execute their computations. In a synchronous computation system, a process already has knowledge about all incoming messages while it needs to perform its computations throughout the entire operational period [23].

3.4 The Proposed Algorithm for Tree Topology

The donor nodes nodes function as underutilized resources while the deficient nodes function as overburdened resources. The resource distribution and discovery problem functions as a load distribution problem. We are using modified diffusion based approach where resources aie assumed to be divisible only integrally and resource ¬surplus nodes share their resources in such a way so that they themselves don't become resource-deficient. Bertsekas and Tsitsiklis showed that an N processor system with total system load L unevenly distributed across the system diffusion approach causes each processor's causes each processor's load to change to LjN. A processor which operates at either high or low capacity performs load balancing with its neighboring processors through simultaneous load distribution.

3.4.1 Synchronizing Mechanism of Balancing Domains

The load balancing process gets activated by resource surplus nodes when they interact with their closest nearby nodes. The balancing domain of a resource surplus node includes its immediate neighbors [l](because resource distribution occurs within one hop) and itself. The balancing domain of simultaneous invokers can either share common boundaries or maintain complete separation. The nodes run resource distribution processes which operate in distinct domains that include single balancing areas and multiple overlapping balancing domains. Resource distribution occurs independently for processors working in separate balancing domains whereas overlapping domain processors execute resource distribution through synchronized operations.

The system uses this method to guarantee that only one of the two overlapping balancing domains operates resource distribution within its area. The resource surplus nodes start the donor initiated algorithm which functions as the main process of this system. The donor nodes send an I_AM_STARTING message to their resource deficient immediate neighbors after they complete the spanning tree construction phase of resource distribution. The resource deficient neighbors send an OK message back after they receive the I_AM_STARTING message, which includes their resource requirements and the number of resource surplus nodes, which they have in their area.

The system needs to distribute resources from only one of its overlapping balancing domains because resource deficient nodes which exist in multiple domains need to send OK messages that falsely report their resource status. A resource surplus node will start giving resources only when it has received OK message from all of its resource deficient neighbors only one of the overlapping domain can start the resource distribution. The donor node will start serving requests after it receives OK messages from all its deficient neighbors by following the order of decreasing priority among the deficient nodes. The number of resource surplus neighbors that a deficient node possesses determines its priority for resource distribution. The donor will serve the deficient node with greater request because it needs to resolve the tie between two deficient node priorities.

To have an intuition of the scheme consider the example below.

See fig 3.1. Node i and node j are resource surplus. Balancing domain of node i consists of node { *a, b,* c, *d* and i}. Balancing domain of node j consists of node { c, *e,j,g* and *j}.* Node i and node j send *I_AM_STARTING* message to its immediate neighbors within 1 hop i.e to nodes *a,.b,c,d* and to nodes c,e,j,and *g,* telling them how much resources they can give to them. Resource deficient nodes upon receiving *LAM_STARTING* from all the adjacent resource surplus nodes send *OK* message. Now in order to synchronize the two balancing domains, resource deficient neighbors can follow the greedy approach.

Policy 1) Resource deficient nodes send reply to *I_AM_STARTING* by sending *OK* message along with their resource requirements to all the neighboring resource surplus nodes. Now consider the case that node c after getting *I_AM_STARTING* message from node i and node j sends *OK* message to both the nodes i and j. In that case, both nodes i and j will start giving resources to their immediate neighbors but that resource distribution will be inefficient because, resource requirement of node c may be denied or accepted by both the balancing domains. By synchronizing the balancing domains we

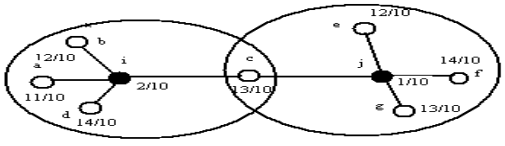

Figure 3.1: tree topology - distributing resources

intend to achieve better resource distribution. This can be done if we allow the overlapping balancing domains to distribute resources one after the other synchronously. For this we can adopt the second policy.

Policy 2) Resource deficient send *OK* message to the resources surplus node which claims to have maximum resources, following the greedy approach. So after one of the balancing domain distributes resources and give resources to node c, node c sends *OK* with modified resource requirement to the next balancing domain. In this way by synchronizing the balancing domain resource deficient nodes gets resources efficiently. However, if we use the synchronization policy 2 over arbitrary topology we may face the problem of deadlock because of circular wait condition. For instance, consider the figure 3:2,

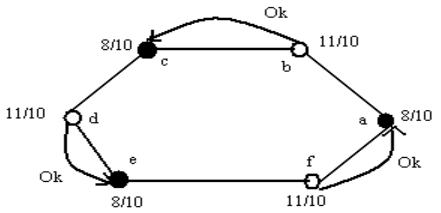

fig: Deadlocked Situation

Figure 3.2 Resource Distribution

here node *a,c* and e are donor nodes having surplus resources of 2,2 and 2 units respectively. Deficient nodes are nodes *b, d* and *f.* Node *b* upon receiving LA~LSTARTING message from donor nodes *a* and c sends OK message to node c. Similarly, node *d* and *f* sends OK message to node e and *a* respectively. In this situation a deadlock has occurred as donor nodes *a,* c and e are circularly waiting for deficient nodes *b, d* and *f* to send OK message. So we use tree topology as it does not contain any cycles and thus we can avoid circular wait condition, hence the deadlock condition.

3.5 Proposed Algorithm for Tree Topology

Problem: Each node i is having some capacity (resource owned by the node) C_i and some requirement (resource required by the node) R_i. 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. $R=\{i: R_i > C_i\}$. Let S be the set of all the nodes whose capacity is more than their requirement i.e.

$S=\{ i : R_i < C_i \}$. Let $T = \{t_{ij} \}$ (where i=l, n and j=l, n) be the transfer matrix denoting the amount of resource which is transferred. $t_{ij} < 0$ means node n_i will receive T_{ij} units of resources from node n_j. $t_{ij} > 0$ means node n_i 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, $\Sigma ((r_i - c_i) + \Sigma t_{ij})$ under the constraint

$$ni \varepsilon R \qquad nj \varepsilon S$$

that $\forall$ *ni* ε *S,*$((c_i - r_i) - \Sigma t_{ij}) >= 0$ i.e. we have to satisfy

$$\forall\ nj \varepsilon R$$

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.

Input of the Algorithm*:* The algorithm which we present in this paper has two distinct operational stages. The first phase requires the creation of a spanning tree which will connect all points in a network that has no specific design. The output generated by the initial phase of the algorithm functions as the input for its second phase which implements resource distribution through the spanning tree. The standard spanning tree algorithm will be applied to create a spanning tree which will connect all points in the network. The relationship between leader election processes and spanning tree construction algorithms shows an equivalent connection according to reference [6]. The second part of the algorithm is currently under process of evaluation. The network is assumed to have a spanning tree which all vertex nodes know each of their direct neighbors together with their resource needs and their capacity limits while they maintain complete knowledge of their resource needs and their capacity limits. Vertex node i has two resource parameters which are resource requirement Ri and capacity Cj. The nodes in the network establish connections which create a tree structure. Each vertex node i knows which node are its children (denoted by Child(i)) and which node is its parent (denoted by Parent (i)).

Output of the Algorithm: The deficient node i has knowledge about T which contains the resource amounts that it will receive from its neighboring nodes. The value of tij becomes negative when

node i receives resources from node j while it becomes positive when node i delivers resources to the same node. The immediate donor neighbors will provide each deficient node with information about the resource amounts which they will receive. The donor will provide each of its immediate deficient neighbors with information about the resource amounts which it will deliver to them.

3.5.1Pseudo code of the Resource Distribution Phase of the Proposed Algorithm for 1 Hop for Tree 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 deficient nodes or donor nodes.
}

When a node receives TERMINATE_PHASE1-MSG

{
If the receiving node is a donor node
It sends I-AM-LSTARTING.-MSG with the amount of resources they can give.
If a donor node has no resource deficient neighbor nodes send I-AM.-DONE-MSG to its parent.
Else if the receiving node is a deficient node
Forward TERMINATE_PHASE1-MSG to all of its children.
}

When a node receives I-AM-STARTING-MSG

{
Receiving node marks I-AM-STARTING_MSG has been received.
If a Receiving node received I-AM-STARTING_MSG from all resource donor nodes
{
Find the max bidder i.e. donor node which has maximum resources to give. Send OK_MSG to the max bidder donor node with the resource required and number of neighboring donor nodes it has.
} else {
}
Wait for all I-AM-STARTING_MSG to arrive.
}

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 OK_MSG 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 its OICMSG contains some request for resources for resources (>= 0)
{

Send GIVE-MSG with resources given in decreasing order of priority.
}
If receiving node has no child
{
Send I-AM-DONE-MSG to its 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. If Receiver node's resource request has been satisfied completely
{
Send OK_MSG to all donor nodes to which Receiver node had deferred sending OK_MSG.
Mark reply to I _ AM_STARTING_MSG i.e. OK_MSG has been sent.

}
else {
Mark number of number of donor nodes has decreased by one.
Mark the sender node is no longer a donor node.
Mark resources available at sender node = O.
If Receiver node has received GIVE_MSG from all the (original)
neighboring donor nodes

{
Send OK-MSG to the donor node having the lowest ID
among the remaining donor nodes corresponding to whose request OK_MSG has not been sent.
Mark reply to I_AM_STARTING-MSG i.e. OK_MSG has been sent.
}
else {
Wait for GIVE_MSG to arrive from all the neighboring donor nodes.
}
If Receiver node is not ROOT and has received all GIVE_ MSG s and
all I_AM_ DONE-MSG to its parent
}
}

When a node receives I_ AM_DONE_MSG
{
Mark **I_ AM_DONE_MSG** has been received from the sender node.
If Receiver node has received **I_ AM_DONE_MSG** from all the children nodes
{
If Receiver node is ROOT node
{
Send TERMINAT_MSG to all the children
}
else
{
.
If Receiver node was originally resource deficient.
{
Send **I_ AM_DONE_MSG** to parent if all GIVE_MSGs have been received from all neighboring donor nodes.
Otherwise Wait for all GIVE_ MSG s to arrive.
}
Else if Receiver node was originally donor node

{
Send **I_ AM_DONE_MSG** to parent if all GIVE-MSG have been sent to neighboring deficient nodes.
Otherwise Wait till all GIVE-MSG have been sent to neighboring deficient nodes.
}
}
Else
{
If **I_AM_DONE-MSG** not received from all children wait till all the messages have been received.
}
}
When a node receives TERMIN ATE_MSG
{
Send TERMINATE.MSG to all its children.
}

3.5.2 Complexity Analysis of the Resource Distribution Phase of the Proposed 1- Hop Algorithm for Tree topology

Message Complexity: As on each edge between donor node and deficient node of the spanning tree, constant number of messages pass. To be more precise during resource distribution at most 5 messages pass over each edge (1*I-AM_STARTING* message, 1 *OK* message and 1 *GIVE* message, 1 I_*AM_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(N) as all the resources are distributed along the edges of spanning tree.

Bit Complexity: Each message has O(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 balancing domain takes 0(1) time. Overlapping balancing domains in which resource-surplus nodes are adjacent also share.

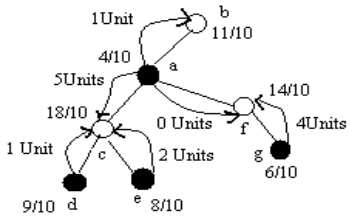

Figure 3.3: Optimal distribution

Example where non-optimal distribution is yielded by our algorithm. Observe that requirement of 4 units of node c remains unfulfilled even though 4 units of resources are available at node *g*. Value *(a/b)* beside node i indicates that node i has *a* units of requirement 'and *b* units available on it 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 $O(K_0)$ time where K_0 is the maximum number of overlapping domains. Also O(h) is the time to detect termination. Thus the overall time complexity of resource distribution phase is O(h) where h is the height of the spanning tree.

3.5.3 Example to show that minimization of unfulfilled request is non-trivial

We are minimizing the amount of unfulfilled request of deficient nodes. For this the proposed algorithm for 1 Hop uses local knowledge to distribute resources. We show that this problem is non trivial by giving an example where the distribution achieved on the basis of local knowledge is poorer than the distribution achieved by the ILP solution. Hence we aim to propose algorithms which give distribution close ILP solution. When deficient nodes fall in overlapping domain of two or more donor nodes, they follow a greedy approach and send *OK* message to only that donor node which claims to have the maximum amount of resources which may not be a correct approach. Also when a donor node receives *OK* message from all of its deficient neighbors, it sends *GIVE* message to the deficient nodes in the decreasing order of priority. Priority of a deficient node is defined by the number resource surplus node it has. Consider the following situation, fig 3.3. Observe that node *f* falls in overlapping domain of donor node *a* and donor node *b.* Node *f* upon receiving *I_Am_Starting* message from donor nodes *a* and *b.* It follows a greedy approach and sends *OK* message to node *a* as it has more resources. Node *f* did not know that node *a* has more deficient neighbors. Also when donor *a* distributes resources it satisfies the request in the order of node *a,f* and c ill the increasing order of priority. However, while distributing resources node *a* does not know that though deficient node c has more neighboring donor nodes,

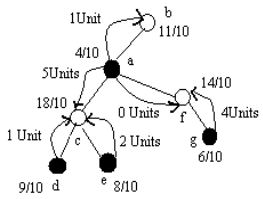

Figure 3.4 LLP formulation

Figure 3.4: Optimal distribution obtained by solution of corresponding ILP formulation. Observe that request of node c is completely fulfilled. Value *(a/b)* beside node i indicates that node i has *a* units of requirement and *b* units available on it they have very less resources available. This is in contrast to the node *f* which has lesser donor nodes but they are have more resources. As donor node *a* distributes based on local knowledge it does not distribute resources optimally and eventually when the. algorithm terminates request of c remains unsatisfied even though there are resources available on node *g.* The value of the metric corresponding to the distribution yielded by our algorithm is 4 while the value of the metric corresponding to the solution yielded by the corresponding ILP formulation of the problem is O. Note that, we had set our objective to minimize the value of metric.

4. RESULTS:

The Tree Topology-Based Resource Allocation and Discovery Algorithm (TTRADA) was evaluated against two baseline approaches—Flat Probabilistic Allocation and Random Walk Discovery—over increasing node densities in a one-hop network (100 to 1000 nodes).

The success rate of TTRADA showed better results than both of its competing systems. At 100 nodes, TTRADA achieved a 97.8% success rate, compared to 92.5% for the flat probabilistic method and 89.6% for random walk. The TTRADA system demonstrated high success rates which reached 90.4% at 1000 nodes because it handled network congestion and contention better than both the Flat Probabilistic system, which achieved 79.7%, and the Random Walk system, which reached 71.3% success. The hierarchical coordination system of TTRADA leads to better conflict resolution because it lessens duplicate requests made by users.

The system TTRADA showed effective performance regarding its latency. The average latency grew moderately from 11 ms at 100 nodes to 27 ms at 1000 nodes. Flat Probabilistic and Random Walk showed greater latency increases because their unstructured coordination systems caused delays for resources to reach their destinations, which resulted in multiple failed attempts to connect.

The TTRADA system uses a virtual tree-based structure which enhances resource discovery and allocation success rates while decreasing response times, which makes the system better suited for expanding networks.

The document presents results from the Tree Topology-Based Resource Allocation and Discovery Algorithm (TTRADA) testing in a one-

hop anonymous network scenario, which includes graphical data and tabular data.

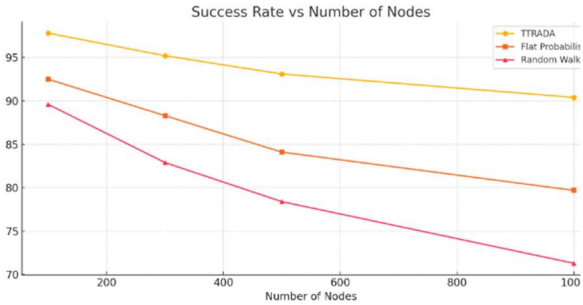

Figure1: Success Rate Vs Number of Nodes

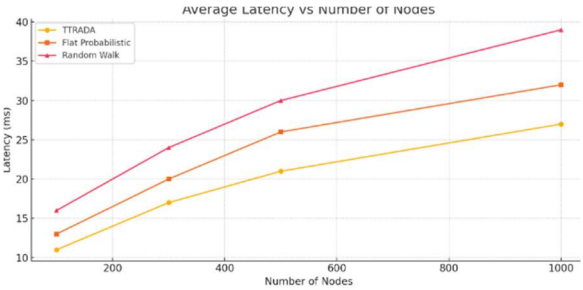

Figure 2 : Average Latency Vs Number of Nodes

Table: Resource allocation

Nodes	TTRDA %	Flat Prob %	Random walk %	TTRADA	Flat prob (Ms)	Radom walk (Ms)
100	97.8	92.5	89.6	11	13	16
300	95.2	88.3	82.9	17	20	24
500	93.1	84.1	78.4	21	26	30
1000	90.4	79.7	71.3	27	32	39

5. CONCLUSION:

We created a general resource discovery and allocation problem to solve the virtual cache allocation challenge of this paper. The presented formulation is universal because we created our heuristics to handle all types of static resource distribution instead of base cache distribution rules. we introduced the foundational model together with the system framework for virtual caching. The study introduced new resource discovery methods together with various load distribution techniques. The existing load distribution methods and resource discovery methods failed to solve our problem which aimed to reduce unfulfilled requests from deficient nodes.

The three proposed heuristics provide solutions for our problem. The first heuristic establishes resource distribution across tree networks to eliminate potential deadlock-inducing cycles. The second heuristic improves upon the first by implementing request sequence numbers for deadlock resolution while maintaining resource distribution across all network components. The sequence number functions as the exclusive identification system for the sender node. The complete complexity analysis of our algorithms demonstrates their performance characteristics. The distribution of resources needed to achieve optimal results requires nontrivial efforts to drop unfulfilled requests from deficient nodes.

Future work: The Tree Topology-Based Resource Allocation and Discovery Algorithm (TTRADA) demonstrates effective resource management capabilities for one-hop anonymous networks through its structured resource management approach but needs additional research and development work in various fields. The primary research area focuses on developing enhanced methods for stable tree construction which can handle extreme changes throughout the entire system. The implementation of mobile ad hoc networks (MANETs) and dynamic Internet of Things (IoT) clusters requires systems that can handle continuous node movements among joining, leaving, and state-changing activities. The system requires protocols that can create tree structures with minimal resource use but maintain operational ability during tree structure changes. The research should examine how local repair methods together with timed soft-state updates can create more reliable systems.

The current version of TTRADA operates under the assumption that all nodes will exhibit identical operational patterns together with equal processing capabilities. The actual situation shows that different nodes will display distinct levels of available resources together with different energy capacities and operational dependability. The algorithm's upcoming versions will use heterogeneous node roles which will position high-capacity nodes as tree structure anchors and coordinators to enhance load distribution and decision-making capabilities.

The system needs machine learning model integration to enhance its tree construction process together with role assignment and resource prediction capabilities. Nodes can improve their strategies through reinforcement learning or federated learning methods which enable them to

analyze their past allocations. nodes need to adapt their methods according to current network conditions and original resource requirements.

Security emerges as a critical domain which requires development during upcoming research phases. The anonymous and decentralized system of TTRADA creates a security vulnerability which enables attackers to perform malicious actions and make false resource claims. The system allows integration of lightweight trust models and reputation systems which enable honest user participation while maintaining system performance.

The testing process requires actual deployment in real environments to validate results which go beyond simulation testing. The TTRADA implementation on embedded devices such as Raspberry Pi and ESP32 will enable testing under actual conditions of interference and packet loss and energy limitations which will generate important findings for future smart city and industrial Internet of Things and disaster recovery network deployments.

REFERENCES:

[1]Xu, Chengzhong, et al. "Nearest-neighbor algorithms for load-balancing in parallel computers." *Concurrency: Practice and Experience* 7.7 (1995): 707-736...

[2] Wang, Yung-Terng. "Load sharing in distributed systems." *IEEE Transactions on computers* 100.3 (1985): 204-217..

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

[4] Chandra, J. (2002). *Analytic and simulation studies on effect of distribution of caches in networks, M. Tech* (Doctoral dissertation, Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, 721 302, India).

[5] Mohapatra, P. K. (2000). Fully Sequential and Distributed Dynamic Algorithms for Minimum Spanning Trees. *arXiv preprint cs/0002005*.

[6] Santoro, N. (1984). On the message complexity of distributed problems. *International journal of computer & information sciences*, *13*(3), 131-147.

[7] Casavant, T. L., & Kuhl, J. G. (2002). A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on software engineering*, *14*(2), 141-154.

[8] Bubendorfer, K. P. (1996). *Resource based policies for load distribution* (Doctoral dissertation, Master's thesis, Victoria University of Wellington) ...

[9] Goscinski, A. (1991). Distributed operating systems: The logical design. *Addison-Wesley Publishing Company*.

[10] Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems: Principles and paradigms* (2nd ed.). Prentice Hall.

[11] Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). *Distributed systems: Concepts and design* (5th ed.). Addison-Wesley.

[12] Bertsekas, D. P., & Gallager, R. (1992). *Data networks* (2nd ed.). Prentice Hall.

[13] Kurose, J. F., & Ross, K. W. (2017). *Computer networking: A top-down approach* (7th ed.). Pearson.

[14] Shenker, S., Clark, D., Estrin, D., & Herzog, S. (1996). Pricing in computer networks: Reshaping the research agenda. *ACM SIGCOMM Computer Communication Review*, 26(2), 19–43.

[15] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2003). Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1), 17–32.

[16] Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001). A scalable content-addressable network. *ACM SIGCOMM Computer Communication Review*, 31(4), 161–172.

[17] Rowstron, A., & Druschel, P. (2001). Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms* (pp. 329–350).

[18] Heinzelman, W. R., Chandrakasan, A., & Balakrishnan, H. (2002). An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4), 660–670.

[19] Gupta, I., Riordan, D., & Agrawal, S. (2001). Evaluating the scalability of epidemic protocols. *Proceedings of the IEEE International Conference on Distributed Computing Systems*, 283–290.

[20] Kleinberg, J. (2000). The small-world phenomenon: An algorithmic perspective. *Proceedings of the ACM Symposium on Theory of Computing*, 163–170.

[21] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., & Schmidt, R. (2003). P-Grid: A self-organizing structured P2P system. *ACM SIGMOD Record*, 32(3), 29–33.

[22] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., & Kubiatowicz, J. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 41–53.

[23] Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4), 63–74.

.