Locating Object Efficiently in a Distributed Computing System using Ant Colony Optimisation

Jiabin Li\textsuperscript{1} and Chun Che Fung\textsuperscript{2}

\textsuperscript{1,2}Jiabin Li & Chun Che Fung, School of Information Technology, Murdoch University, Murdoch, Western Australia
e-mail: (j.li, l.fung)@ieee.org

Abstract—Digital Ecosystems reply on efficient computing and communication infrastructures. One way to improve computation efficiency is to utilise distributed computing systems. In an object-based distributed system, the use of location-independent naming scheme can improve the system’s transparency, scalability and reliability. Names however need to be resolved prior to pass messages between the objects. This paper reports the use of a distributed Ant Colony Optimisation algorithms (ACO) to improve the efficiency of searching objects in a distributed computing system. The ACO algorithm is designed for an Adaptive RandoMised Structured search network termed ARMS. The approach provides name resolution by forwarding a query through neighbouring nodes. The performance of ARMS is compared to Chord, a well-known structured network. Simulation studies have shown ARMS is superior to Chord as ARMS requires a shorter path in query forwarding.

Index Terms—object-based distributed systems, distributed searching algorithm, randomised structured network, naming models.

I. INTRODUCTION

Objet-Oriented Programming (OOP) model has become the most popular approach for the development of applications in the digital ecosystem. Due to the increasing need to handle more and more applications and huge amount of data, this has led to active research on the design of more efficient execution of this computational model in a distributed environment. The OOP model is based on the simple concept of objects communicating with one another. An object includes a concept of state and methods. Synchronisation between the objects is therefore intrinsic. On the other hand, the concept of memory is distributed instead of being centralised and shared by the objects. This suggests the implementation could be made in a highly distributed system execution platform with an efficient communication structure.

As far as the access of objects is concerned, every object is referenced through its symbolic name. That name is in ideally to be independent of the physical. While this supports transparency, scalability, object mobility and reliability, the resolution of the physical location relating to the symbolic name may incur a loss of computational efficiency.

A fixed name server is commonly used to provide name translation for distributed systems due to its simplicity. However, such server offers poor scalability, a possible single point of failure, and provides a bottleneck in the system performance. Consequently, peer models [1-4, 6-13] has been widely studied as an alternative that provides scalable and decentralized name translation for large-scale distributed systems. In a peer model, every node takes the responsibility of answering messages that query the node’s local content. The node also has to delegate a message to its neighbours if the node cannot answer the query message. An overlay search network is therefore required to allow peer nodes to establish virtual links between other nodes for query delegation. In general, there are two common approaches to construct a naming system that provides dynamic name translation. They are unstructured model and structured model based on the design of the overlay search network.

Structured search networks [1-4] facilitate load balancing and efficient query routing. In such models, both objects and nodes are systematically organised such that objects are uniformly distributed to each node in the network. In the structured search networks, the object placement policy is tightly controlled. Hence, this facilitates the process of routing the query in locating the target object. This leads to the advantage such that the search step is bound to $O(\log N)$ with high probability where $N$ is the size of the network as stated in [3]. However, such tightly controlled, uniform object placement destroys the flexibility in object placement and related objects are not necessarily located close-by. This is known as the locality issue [5].

Unstructured search networks [6-8] imply neither an object placement policy nor an overlay structure. The nodes are free to choose their neighbours. The features of an unstructured network include support for locality due to flexibility of object placement, resilience to topology change and support for object migration. Routing efficiency and scalability are the main weaknesses of these networks.

In this paper, the design of an Adaptive RandoMised Structured (ARMS) is reported. The design is aimed at improving search efficiency and flexibility of existing search networks. ARMS is based on a randomised structured network [9-13] that provides the flexibility of neighbour selection. The proposed model utilises a distributed Ant Colony Optimisation algorithm (ACO) to improve the ability of path exploration. The design of the model is described in Section III and the location protocol is presented in Section IV. Section V gives the test results based on different system parameters and the performance comparison between Chord and ARMS using simulation. Finally, Section VI concludes the paper with discussion on future work.

II. RELATED WORK

Recent studies of location lookup for large-scale, dynamic networks focus on the use of distributed models, par-
ticularly in the area of peer-to-peer (P2P) file sharing systems [1-4, 6-8]. In a P2P system, each node has similar functionality and shares the responsibility of object binding and location lookup with other nodes. Because both control and data are distributed among the nodes, name services can be performed locally, leading to better scalability and efficiency. To find a node that is responsible for the particular symbolic name or key, the requesting node queries other nodes by sending messages. The two models - structured and unstructured search networks, for building distributed name translation system are discussed below.

A. Unstructured search networks

The unstructured model is a popular approach to provide distributed location lookup. They imply neither a centralized control nor any structured data organization is deployed. Without the complete knowledge of the network structure, flooding is a typical approach to locate an object in the network. In this approach, a node queries all its neighbors within a certain radius [6] in order to locate the target object. This approach is extremely costly and leads to poor scalability and long response time. Furthermore, flooding causes messages to be received by all nodes in the neighbourhood. These broadcast messages do not improve the probability of finding the requested object but they can cause network congestions and block other useful messages.

Alternative protocols such as probabilistic flooding [7, 8], and random walkers [14] have been proposed to improve scalability and routing efficiency. In probabilistic flooding, a subset of the neighbors is selected to forward each query message. However, this model still gives poor performance due to the duplicated messages. Highly compressed data structures like Scalable Query Routing (SQR) [15] has been proposed to reduce the number of query messages being propagated to the neighbors.

Random walk models have been proposed in [8, 14] where only one neighbor is chosen at each cycle of query propagation. This significantly cuts down the total number of queries required. The weakness of the scheme is that the expected search path is extremely long due to the randomness of the algorithm. A better approach is to use N random paths in the search [14]. However, the choice of N has significant impact on the efficiency because the use of excessive random walks leads to the same problem in flooding.

B. Structured search networks

Some better known structured networks are CAN (content addressable network) [1], Chord [2], Pastry [3] and Tapestry [4]. In these systems, namespace is structurally organized and tightly controlled. The namespace is uniformly assigned to each node in the network and hence the model ensures that the network load is balanced among the nodes. A different data placement policy may lead to a different routing scheme and thereby affects the performance of the location lookup process. For instance, the namespace structure of CAN is based on a d-dimensional Cartesian coordinate scheme. Each node has a constant number of neighbors O(d) and the expected querying path lengths are O(dN^{1/d}), where N is the number of nodes in the network. Chord uses a one-dimensional circular structure with the expected number of neighbors to be O(logN), and path lengths of O(logN).

Despite the popularity of the structured model, the model has several issues relating to the tightly controlled data placement. First, objects are not randomly located but are placed at specific locations. This makes object migration inconvenient but it is essential given the need for load shedding in distributed computing environments. Secondly, structured networks assume objects have uniform demands. In other words, the objects are assumed to send or receive a similar number of queries. However, most practical systems [16, 17] follow a power-law distribution, that is, only a few objects have most of the incoming requests. Kalogeraki et al in [18] point out that non-uniform traffic can cause query hotspots and routing hotspots in structured networks. Query hotspots are caused due to the result of some popular objects that are frequently being requested by other objects. This problem can be resolved by caching the popular objects. Routing hotspots are caused by unbalanced requests sent by some node. This issue is harder to handle due to the use of fixed routing used in most structured networks. Routing hotspots will cause link congestions. Finally, the flat namespace used in structured networks destroys object locality [5]; related objects may not be located closely in structured networks.

III. ARMS

ARMS is a distributed naming model that aims at providing efficient and scalable naming services for an object-based distributed system. The heart of the model is a randomised structured network. The network takes the advantages of structured networks including availability, the balanced routing state and the scalable searching path. On the other hand, ARMS allows each node to choose its neighbours with the greater flexibility compared to a structured network. A structured network requires a deterministic neighbour selection scheme that is merely based on the distance of two nodes on the virtual topology. Such selection scheme largely assumes a homogeneous environment, where nodes have equal power and workloads. This is an unrealistic assumption as most practical systems produce workloads that follow a power-law distribution [16, 17]. As a result, randomised networks [9-13] offer more flexibility on neighbour selection so that each node is able to exploit the full potential of its neighbours. Such flexibility also provides a platform for adaptability. Structured networks limit the ability of a node to explore better routes due to their rigid organisation and preliminary routing. Because the state of a practical network is constantly changing and possibly chaotic, it is desired to allow each node to decide a route based on the current status of the network in order to achieve better utilisation. Consequently, a flexible network is crucial to the routing performance.

The flexibility of randomised structured networks enables nodes to exploit the heterogeneity of neighbours. However the query forwarding scheme of these networks either is too basic or requires a periodic process to update the state of the forwarding table like RASTER [13]. More importantly, these networks have not considered a mecha-
nism to support path exploration as their query forwarding schemes are mostly based on greedy routing. A problem of many such algorithms is that they mostly fail to find the global optimal solution although they offer faster computation than other techniques. To address this issue, ARMS uses an adaptive forwarding mechanism designed using a distributed Ant Colony Optimisation algorithm in order to support neighbour exploitation and path exploration. Compared to RASTER [13], ARMS requires just a single message to update the forwarding table of nodes that reside on the best route between the initiator and the queried target by piggybacking updating information in the reply message.

The Ant Colony Optimisation algorithm (ACO) is a probabilistic technique for solving optimisation problems, which can be abstracted to finding the best path on a graph [19, 20]. It is inspired by the behaviour of some ant species in searching paths from the colony to food. In analogy to biology, artificial ants deposit a substance termed pheromone on the trail in order to encourage other members to follow. The importance of pheromone is to stop ants wondering at random but to instead utilise and reinforce the path that eventually leads to the food source. Over time, the pheromone starts to evaporate and thus the trail loses its concentration. The more frequently ants will travel down the trail, the higher concentration the trail can have and so the longer time the process of the pheromone evaporation takes. Through this, the pheromone of good or promising trails will be strengthened and the bad ones will be weakened.

Ants select a path via a stochastic mechanism that is based on the values of the pheromone and the heuristic information. In ARMS, the use of the heuristic information is twofold. First of all, the constructive heuristic can provide guidance to ants at the early stage of the search. Secondly, the heuristic information can be changed during computation in order to adapt to the changing state of the network. That is, it helps ants to choose alternative paths in case the good solutions known previously become an obstacle due to the dynamic nature of the network. Such mechanism is vital for support for path exploration in a dynamic environment. In addition, parallel ants are used initially in order to build a path quickly. Once such path is found, other ants are likely to follow the path as result of pheromone attraction. Consequently, ARMS can effectively reduce the network traffic which is the key issue of flooding-based approaches while it still keeps the resilience of flooding.

IV. ARCHITECTURE OF ARMS

We use Chord [2] to demonstrate the concept of ARMS and its implementation in a practical structured system. This section describes the architecture of ARMS and its location protocol.

A. Network topology and forwarding table

Chord nodes are arranged in a ring topology as illustrated in Fig. 1. Each node has an identifier that is produced by a consistent hash function as similar to Chord [2]. Such identifier is location-independent and thus it is transparent to the structure of the network. Every object is allocated to one of the keys on the ring space. Every node is responsible to only a subset of keys on the ring space in order to balance the state of the routing table across nodes. The key of an object is stored in a node that has the identifier immediately followed that key. Such node is also termed the successor of the key [2]. With Fig. 1, for instance, Node 1 is the successor of Key 1 and Node 3 is the successor of Key 2 and 3. The importance of the consistent hash function is its support for parallel naming assignment. The performance of the object oriented model relies on the efficiency of object communication and object creation. Profiling study shows that object creation is concurrent and thus the parallel naming scheme is well suited to provide name binding for an object-based distributed system.

![Fig. 1: An illustration of Chord ring adapted from [2].](image)

In Chord, a query is forwarded to the neighbour that has closest node identifier to the object identifier or key at the ring space. The efficiency of the query forwarding relies on the design of the forwarding table at every node. For m bits identifiers, each Chord node maintains an m-level forwarding table. The forwarding table stores a pointer termed finger that is pointing to the successor of the key:

\[(n + 2^{k-1}) \leq k \leq m, \quad (1)\]

where n is the node identifier; k is the k\textsuperscript{th} level of the forwarding table; and m is the length of identifier.

The use of fingers is to accelerate the key search as the distance from the next forwarding node to the target node is at most half the distance from the current node to the target. Thus the number of forwarding nodes necessary will be O(logN) with high probability. Detailed proof can be found in [2].

While there is only one node at each level of Chord’s forwarding table, a node is allowed to add new neighbours to the k\textsuperscript{th} level of ARMS table when their node identifiers are belonged to the range of \(2^{k-1}\) and \(2^k\). Although such design breaks the load balance of Chord’s forwarding table, the flexibility of neighbour selection can improve the routing efficiency and latency resiliency. The location protocol is described in the following section.

\[\text{lookup(key)}\]
\[\quad \text{if key} \in (\text{this.id}, \text{successor.id})\]
\[\quad \text{forward the query to successor};\]
else
    \( C = \{\} \); 

for \( i = m \) downto 1 

for all nodes \( n' \) in \( \text{finger}[i] \) 

if \( n'.id \in (n.id, \text{key}) \) 

add \( n' \) to \( C \); 

if this is the initiator of the query 

for \( j = 1 \) to \( q \) 

choose \( n \) from \( C \) with probability \( p_q \) given by Eq. (2); 

forward the query to \( n \); 

else 

choose \( n \) from \( C \) with probability \( p_q \) given by Eq. (2); 

forward the query to \( n \); 

Fig. 2: The location algorithm of ARM5. 

B. Location algorithm 

Fig. 2 depicts the algorithm of query forwarding in ARM5. Upon name lookup, a node first checks whether the request key lies at its successor. If it is found, the query is forwarded to the successor. Otherwise, a node searches its forwarding table for the next node to inquiry. If it is the initiator of the query, the node will forward the query to \( q \) neighbours in order to improve the speed of search. Or else, the node will propagate the query to one of its neighbour in order to reduce the network traffic. In contrast to finding the “closest” neighbour, as required in conventional structured networks, ARM5 node looks for the “best” one amongst neighbours that are resided at the same level as the target in the forwarding table. The best forwarding neighbour is determined through the probability process [19]: 

\[
p_q = \frac{\tau_i^e \cdot \eta_i^\theta}{\sum_{\alpha \in N(s^p)} \tau_{\alpha}^e \cdot \eta_{\alpha}^\theta} \quad \text{if} \quad C_0 \in N(s^p), 
\]

where \( \tau_{ij} \) is the pheromone associated with the edge joining node \( i \) and \( j \), \( N(s^p) \) is the set of unvisited neighbours, the parameters \( \alpha \) and \( \beta \) control the relative importance of the pheromone and the heuristic information \( \eta_i \) which is given by: 

\[
\eta_i = \frac{1}{d_{ij}}, 
\]

where \( d_{ij} \) is the distance of identifiers of the node \( i \) and \( j \) on the ring space. 

C. Updating protocol 

Once the target is found, a reply is returned from the target through the forwarding path. Then, nodes on the path update their pheromone: 

\[
\tau = [(1 - \rho) \cdot \tau + \Delta \tau]^{\text{max}} 
\]

where \( \rho \) is the evaporation rate, \( \tau^{\text{max}} \) and \( \tau^{\text{min}} \) are the upper and lower bounds imposed on the pheromone and \( \Delta \tau \) is defined as: 

\[
\Delta \tau = \frac{1}{L} 
\]

where \( L \) is the length of the path. 

Furthermore, in order to encourage the node to select different neighbour, the pheromone of the edge between the nodes \( i \) and \( j \) is locally updated once the query is forwarded to node \( j \): 

\[
\tau = (1 - \psi) \cdot \tau + \phi \cdot \tau_0 
\]

where \( \psi \in (0, 1] \) is the pheromone decay coefficient, and \( \tau_0 \) is the initial value of the pheromone. 

V. SIMULATION ANALYSIS 

We have conducted a series of simulation tests to study the impact of system parameters on the search efficiency and the scaling property of ARM5. The network traffics were produced from traces that were extracted from an execution of four object-oriented benchmark programs. Java [21] was chosen as the object-oriented language in this study due to its popularity and modernity. Originally developed with an objective to provide platform-independent applications, it is now more noted for its extensive use in networked applications. Four separate Java applications were chosen as benchmarks for study. 

• AutoFocus (AF) [22] is a Java-based desktop search engine. It features in Cluster Maps to present the search results. 

• DynamicJava (DJ) [23] is a Java source interpreter aimed at easing the creation of Java programs. 

• ImageJ (IJ) [24] is a Java-implementation image processing and analysis program with the use of multithreading. 

• Rhino (RH) [25] is an open-source implementation of JavaScript written entirely in Java. 

A. Computational model 

The simulation system implemented two types of essential entities, computational nodes and active objects. A computation node is an abstraction of an execution unit, consisting of a data processing unit and a communication unit. An active object is responsible for reproducing the history of execution for a particular software object. For the sake of simplicity, each node ran exactly one object. Furthermore, a trace file contains only object communications profiled at runtime. We have assumed local computation follows a normalized distribution. 

The design of active objects is based on a distributed object computational model termed the Actor model [26]. An actor performs a combination of the following actions in response to incoming messages: 

• the creation of new actors; 

• messages sent to other actors; and 

• updating of its state. 

The last point is related to local computation while a communication model influences the other two points (i) and
(ii). Message passing is asynchronous in such model. In addition, a physical network structure was also included in simulation in order to study the effects of communication delay on the search algorithms. A 2-dimensional grid was implemented because it is one of the most common structures used in cluster computing systems.

B. System Parameters

The parameters investigated here are those that have direct or indirect impact on computation of the probability in Eq. (2) $\alpha$, $\beta$, $\rho$, and $\varphi$. We tested several values for every parameter, as the rest being constant, in a 2D grid simulation network with the dimension of 40 by 40. The default value of the parameters was $\alpha=0.5$, $\beta=0.5$, $\rho=0.5$, $\varphi=0.5$. Furthermore, four simulations for each configuration were performed to obtain mean values. The values experimented were: $\alpha \in \{5, 8, 10, 15, 20, 25, 30\}$; $\beta \in \{2.5, 5, 8, 10, 15, 20, 50\}$; $\rho \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$; and, $\varphi \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

Simulation results are presented in Table 1-4 as a unit of simulation time. The highlighted values are those that have the shortest length of lookup.

The simulation (Table 1) shows for $\alpha$ a monotonic decrease of the lookup path up to $\alpha=20$. Then the average lookup path starts to rise. The other results show that the length of lookup path decreases with the increase of $\beta$. According to Eq. (2), the larger the value of $\beta$, the more impact the heuristic information becomes. Hence, it is desired to design a good heuristic information in order to improve lookup efficiency. Next, tests on the parameters $\rho$ and $\varphi$ show that $\rho$ has the optimum value around 0.4 and $\varphi$ has the optimum value around 0.5.

C. Search path

The routing performance is generally measured by the search path that query messages have to travel to locate the target object. In this section, we present the distribution of the length of the search path for Chord and ARMS in Fig. 3, and the average length of search path with respect to the network size in Fig. 4. The results indicate that the average length of search path is relatively shorter in ARMS due to its ability to exploit the potential of neighbours and explore an alternative path when the network condition is changed. The use of parallel messages and the updating message can improve the speed of searching while the cost is that it results in significantly more messages compared to Chord.

VI. CONCLUSION

This paper introduces a distributed naming model, termed adaptive randomised structured search network (ARMS), for object based distributed systems. The model is based on a randomised structured model that provides the flexibility of neighbour selection. However, the model also incorporates adaptability by using a distributed Ant Colony Optimisation algorithm. Thus, ARMS can provide efficient search as demonstrated by simulation. As shown in simulation, the heuristic information has significant impact on the length of lookup path. Consequently, selection of good heuristic information is crucial for the performance of ARMS.

VII. REFERENCES