Data Management in Distributed Simulation of Complex Systems

Dan Chen, Roland Ewald, Georgios K. Theodoropoulos, and Ton Oguara

Abstract— Distributed simulation has emerged as an important instrument for studying large-scale complex systems. Such systems inherently consist of a large number of components, which operate in a large shared state space interacting with it in highly dynamic and unpredictable ways. Optimising access to the enormous shared data is crucial for achieving efficient simulation executions. This effort involves two major issues, data distribution and data accessing.

In this paper, we discuss the issues of modelling shared data. We have developed a framework for distributed simulation of MAS, which uses a hierarchical infrastructure to manage the shared data and facilitate interoperation amongst agent simulation models. Our framework aims to reduce the cost of accessing shared data by dynamically redistributing shared data in the infrastructure according to the access pattern of the agent simulation models.

Data accesses may take two forms: locating data according to a set of attribute value ranges (Range query) locating a particular state variable from the given identifier (ID query and update). This paper proposes two alternative routing approaches, namely the address-based approach, which locates data according to their address information, and the range-based approach, whose operation is based on looking up attribute value range information along the paths to the destinations. The two algorithms are discussed an analysed in the context of PDES-MAS, a framework for the distributed simulation of multi-agent systems, which uses a hierarchical infrastructure to manage the shared state space. The paper introduces a generic meta-simulation framework which is used to perform a quantitative comparative analysis of the proposed algorithms under various circumstances.

Index Terms— **Complex systems, Distributed Simulation, Data Distribution, Data management**

I. Introduction

The last decade has witnessed an explosion of interest in complex systems, systems which involve dynamic and unpredictable interactions between large numbers of components including software, hardware devices (such as sensors), and social entities (people or collective bodies). Examples of such systems include from traditional embedded systems, to systems controlling critical infrastructures, such as defence, energy, health, transport and telecommunications, to biological systems, to business applications with decision-making capabilities, to social systems and services, such as e-government, e-learning etc. The complexity of such systems renders simulation (agent-based simulation in particular [5]), the only viable method to study their properties and analyse their emergent behaviour.

The application of simulation to ever more complex problems has placed it in the highly computation intensive world with computational requirements far exceeding the capabilities of conventional sequential computer systems. As a result, distributed simulation approaches are increasingly used in the development and analysis of complex systems. Amongst the most influential of these approaches, the Logical Process Paradigm seeks to divide the simulation model into a network of concurrent Logical Processes (LPs), each of which models some object(s) or process(es) in the simulated system. Each LP maintains and processes a portion of the state space of the system and state changes are modelled as time-stamped events in the simulation. In conventional distributed simulations, the shared state is typically small and the processes interact with each other in a small number of well-defined ways. The topology of the simulation is determined by the topology of the simulated system and its decomposition into processes, and is largely static. However, in the case of systems, which operate in complex environments and interact with it in highly dynamic and unpredictable patterns (such as multi-agent systems, battlefield simulations, ecological systems, games, autonomic systems etc), it is often difficult to determine an appropriate simulation topology a priori. In such systems there is a very large set of shared state variables which could, in principle, be accessed or updated by the processes in the model [28]. Encapsulating the shared state in a single process (e.g. via some centralised scheme) introduces a bottleneck, while distributing it all across the LPs (decentralised, event driven scheme) will typically result in frequent all-to-all communication and broadcasting.

In [17] we have proposed an approach to manage the shared data in distributed simulations of multi-agent systems (MAS). The approach is based on the notion of Spheres of Influence (SoI)} and uses a hierarchical simulation infrastructure to

Manuscript received on January 12, 2006. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under Grant BS123456.

Dan Chen, Georgios K. Theodoropoulos and Ton Oguara are with the School of Computer Science, University of Birmingham, Birmingham, B15 2TT UK (phone: +44-121-4143739; fax: +44-121-4144281; e-mail: {d.chen, g.k.theodoropoulos, t.oguara}@cs.bham.ac.uk).

Mike Lees and Brian Logan are with the School of Computer Science and IT, University of Nottingham, UK (email: {mhl, bsl}@cs.nott.ac.uk).

Roland Ewald and Adelinde M. Uhrmacher are with the Department of Computer Science, University of Rostock, Germany (email: {roland.ewald, lin}@informatik.uni-rostock.de).

dynamically decompose and distribute the shared state. It has been realised in the context of a framework for the distributed simulation of multi-agent systems (PDES-MAS).

Shared data management in distributed simulations needs to address two problems, namely data distribution and data accessing. In an effort to address data distribution issue and provide a generic approach for dynamic load management and interest management in distributed simulation of multi-agent system, in [17][18][29][30] we have introduced the notion of Spheres of Influence (*SoI*). In this paper, we describe an adaptive data distribution scheme, where the shared data is dynamically redistributed to meet the ideal *SoI*. The data redistribution algorithm is detailed in [20]. This paper also discusses the issues of modelling shared state and presents alternative modelling schemes. We analyse the tradeoffs of adopting alternative modelling schemes on the execution of distributed simulation of MAS.

Data accesses target both individual data items (ID queries) and selected data items overlapping given query windows (Range queries). The issue becomes more complicated when the value and the physical distribution of data items both are dynamic. Further problems arise when query sources are changing their positions. This paper proposes two candidate algorithms for data accessing in the context of the PDES-MAS framework, namely the address-based and the range-based routing.

The rest of this paper is organized as follows: DS-MAS framework and Data distribution in DS-MAS are introduced in Section II. Shared state modelling is discussed in Section III. Section IV covers the alternative solutions to data accessing and gives qualitative analysis on the advantages and drawbacks of both solutions. Section V presents the benchmark experiments for studying the dynamics of the routing solutions. Related work and background of this study are briefed in Section VI. In Section VII, we conclude with a summary and proposals on future works.

II. DS-MAS Framework and Data Distribution

We have developed a framework for distributed simulation of multi-agent systems (DS-MAS). The framework models a multi-agent system as a network of Logical Processes (LPs). In particular, each agent is modelled as an Agent Logical Process (ALP) [16]. An ALP has both private state and shared state. The private state are maintained within the ALP, while the shared state can be accessed (read or updated) by other LPs. State changing of an ALP that has impact on LPs' state is called an external event of the ALP, and this is represented by an operation on the shared state. The shared state is modelled as a set of Shared State Variables (SSVs). External events are modelled as timestamped messages being exchanged amongst LPs. The shared state and external events are the data managed by the framework.

In the DS-MAS framework interoperations between ALPs are facilitated by a hierarchical simulation infrastructure. The infrastructure is constituted by a network of Communication Logical Processes (CLPs), which is the manager of the shared data and provides common services to the ALPs. The services provided by the CLP tree include: (1) facilitating the construction of the distributed simulation; (2) clustering and interoperating the ALPs; (3) managing shared data and balancing load incurred by accessing the shared state; and (4) facilitating synchronization of the ALPs. Figure 2 illustrates the relationship between an ALP and the CLP tree. Operation of the CLP tree remains transparent to the ALPs during the simulation. The DS-MAS framework provides a software library to the ALPs to interact with the CLP tree through the two interface modules, namely a *SimulationAmbassador* and *AgentAmbassador* module. An ALP issues requests to access shared state variables through the *SimulationAmassador* module which forwards the requests to the server CLP. If the required SSV is not held locally, the server CLP passes the request to its parent CLP to deal with the request. The return data and control messages (e.g. rollback) are conveyed to the ALP via its *AgentAmbassador* module.

Figure 2 gives a schematic view of a CLP, which interacts with other LPs via ports. Ports link the individual LPs together to form the overall DS-MAS simulation system. In this paper, we name the port of a CLP, from which a request on accessing SSV is received, the incoming port. Each CLP is also a router responsible for forwarding an access request to the destination CLP(s) that host the target data. If this CLP needs to forward a request out via other ports, those ports are called outgoing ports. The port is specially designed to maintain the distribution of the values of SSVs in the value space[1] classified by the types of SSVs (also see Section III.A). This paper uses the term "*attribute value range*" to denote a certain range of the values of a set of SSVs associated with a particular attribute (or a set of attributes). The attribute value range can be described as simple as *[Min, Max]* or a more detailed list or other data structure with different "resolutions". The "extent" covered by the attribute value range may vary with different routing algorithms. The distribution concerns SSVs in the local CLP and/or SSVs in remote CLPs, for instance the overall system beyond a port or only the direct neighbours (parent and/or children nodes if any) to the CLP from this port. The distribution of the values gives a panorama of the status of SSVs in the system. No matter what resolution and extent are chosen for an attribute value range, it should always correctly reflect the status of SSVs and can be refreshed once the status changes.

In the context of DS-MAS framework, the *SoI* of an event is defined as the set of shared state read or updated as a consequence of that event [18], which describes the immediate effect of an event. The *SoI* of an agent simulation model over a time interval is the union of the spheres of influence for each event generated by the agent simulation model within the interval. Intersecting the spheres for each event generated by the

[1] For example, we define "x-position" in an extent [0, 100]. Given 100 SSVs of x-position, the values of these SSVs may distribute evenly from 0 to 99, such as (0, 1, …, 99), or concentrate on [50, 51].

agent simulation model gives a partial order over sets of shared state over the interval, which denotes the frequency with which these portions of shared state been accessed. The *SoI* describe the access pattern of the simulation models on the shared state in a given time interval. In more recent work we have realised this approach in the DS-MAS framework [18].

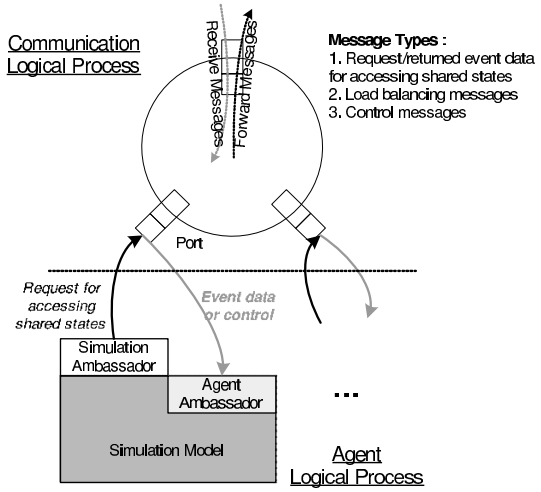

Fig. 1. Relationship between the CLP Tree (Hierarchical Simulation Infrastructure) and ALPs

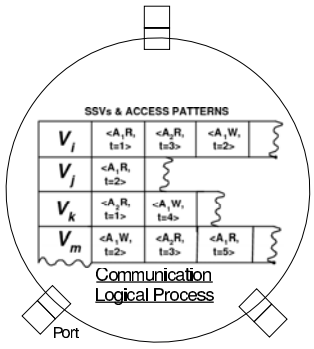

Fig. 2. Communication Logical Process and Ports

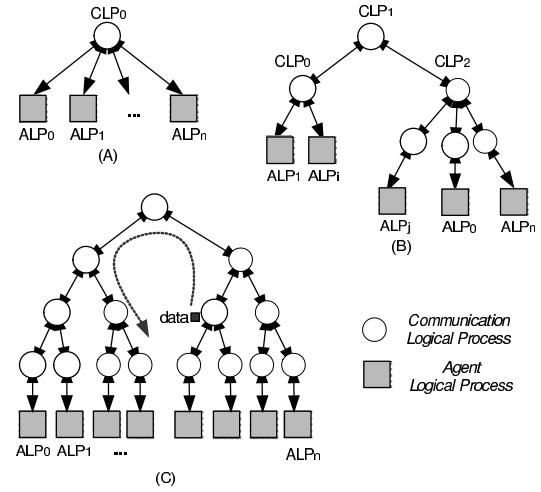

Fig. 3. Illustrating the DS-MAS framework

In [20], we propose that the simulation initially starts with a single centralised CLP marked as CLP_0 (see Figure 3). As the simulation execution proceeds, CLP_0 may become a bottleneck due to the increasing load of managing SSVs. When this happens, CLP_0 is split into multiple CLPs, with each new CLP maintaining a subset of the shared state which is most closely associated (in terms of their spheres of influence) with the ALPs which are below it in the tree. This reduces the load on individual CLPs, and such "decomposition" of congested CLPs naturally leads to the construction of a CLP tree (Figure 3(B)). As the access patterns on the shared state change, so does the configuration of the tree and the distribution of state (i.e., its allocation to CLPs) to reflect the logical topology of the model.

Redistribution can be achieved in a number of ways, such as (1) Moving/merging/splitting CLPs, (2) Moving ALPs and (3) Moving state between CLPs. Considering the design complexity and the cost of manipulating LPs at runtime, in this project we choose to use a fixed tree of CLPs and move SSVs through the tree to achieve redistribution. A simple way to construct a fixed tree is using binary tree structure as it eases searching and manipulating data (Figure 3(C)). Although a CLP tree's topology is fixed, its scale is determined by the number of ALPs involved in the simulation. The CLP trees discussed in this paper are binary trees, thus it is straightforward for us to specify a permanent address to each CLP. Each ALP links to a leaf CLP directly, this leaf CLP is called the ALP's server CLP while the ALP is named a client ALP to this sever CLP.

TABLE I
ILLUSTRATING OBJECTS AND SHARED STATE VARIABLE

	Object Class	Attribute ID	Object Instance	Shared State Variable
Name	"Tile"	"X-pos" of the "Tile" class	A "Tile" with name "Tile123"	The "X-pos" of object "tile123" at time T.
Mapping to C++ concepts	Class "tile"	Member variable "X-pos" in class "tile" (tile::X-pos)	An instance of "tile" (Tile123 = new tile())	The member variable "X-pos" of Tile123 (Tile123->X-pos)
Representation in programming	10	101	3000	<10, Tile123, 101, value of X-pos, T >

III. MODELLING SHARED STATE

Modelling shared state is an important task for manipulating shared data in distributed environment. In the DS-MAS framework, there are four basic notations related to share state: Object Class, Object Instance, Attribute ID and Shared State Variable (SSV). An Object Class defines the conceptual representation of a collection of shared data with same properties. Each object class has the same semantics and common meaning crossing the distributed system. An Attribute ID denotes a characteristic of objects with a particular type, which must be associated with a certain object class. An Object Instance denotes a unique instantiation of an object class, which is created by some ALPs. Object instances can be abbreviated as "objects" in this paper. The attribute values of an object represent the state of this object.

For those objects that can be globally accessed by ALPs, an attribute (or a certain combination of multiple attributes) of each object forms an SSV in the system. The attribute(s) are named as the type of this SSV. An SSV is said to have a simple type if it represents a single attribute of an object or have a composite type in case it describes multiple attributes of an object. An SSV must be associated with a particular object. To precisely describe an SSV, an SSV data item should contain: object class, object ID, attribute ID, value and the timestamp(s) at which it has been updated. Table 1 illustrates the relationship amongst the four terms using the "*TileWorld*" simulation as an example [23][31]. The last row in Figure 4 lists representations of the type, attribute, ID of an object and the ID of an SSV associated with the object[2].

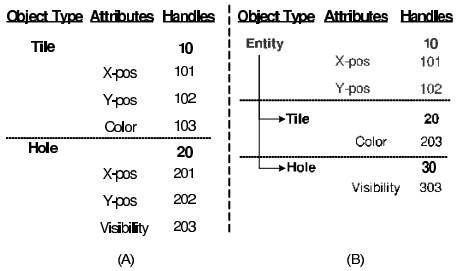

Fig. 4. An Example of Defining Object Classes and Attributes

Figure 4 depicts two examples of defining object classes which adopt a similar scheme to Federation Object Model [20]. Figure 4(A) gives an example of defining two element classes, which means their associated attributes are totally independent; although their attributes are specified the same names or stands for the same abstractions in the physical system. There are no common attributes between element object classes.

As for common attributes of different object classes, we may adopt a scheme as indicated in Figure 4(B). Here we assume that the modeller intends to make no difference between object types for those attributes. Object class "*Tile*" and "*Hole*" (sub-types) can be derived from the same base class "*Entity*" (super-type). By their mature, the common attributes "*X-pos*" and "*Y-pos*" (from the perspective of object class *Tile* or *Hole*) belong to the objects of "*Entity*". Therefore, the two attributes are assigned the same IDs for all sub-type objects of "*Entity*". However, it is impossible to determine whether the attribute ID 101 belongs to a "*Tile*" or a "*Hole*". Sometimes the common attribute will be used in performing range queries. This issue will be discussed later in Section III.C and IV.D.2.

A. *Operations on Shared State*

ALPs can perform a number of operations on the shared state [20]:

1) Requesting the value(s) of one or more attributes with a given timestamp.
2) Updating the value(s) of one or more attributes at a designated simulation time.
3) Adding/Removing one or more attributes from a given timestamp.

Consequently, there exists several different categories of operations on SSVs by both ALPs and CLPs, those are (1) query (read and write (update)), (2) creation and deletion, as well as (3) migration. An LP generates external events using this method.

ALPs are allowed to query SSVs with complex criteria, which may consist of a set of conditions with versatile logics. For example, an agent issues a query like this: the colour of all tiles within the area ($4 \leq X\text{-}pos \leq 6$) and ($5 \leq Y\text{-}pos \leq 7$) OR the area ($X\text{-}pos \leq 3$) and ($Y\text{-}pos \geq 4$) AND the (*X-pos* and *Y-pos*) of "*Hole764*". Nevertheless, a very complex query can always be decomposed into atomic queries, and the returned data in each atomic query will be consolidated and delivered to the ALP via the *AgentAmbassador* according to the logical expression. There are two types of atomic queries, namely Range Query and Shared State Variable Query (SSV Query or ID Query).

A range query requests a set of SSVs of the given type and with value within the given attribute value range. Attribute value range is often used in range queries or describing the distribution of SSVs' value in the system (see Section II). An agent needs to explore some portion of the environment of its interest, which is often independent from the priori knowledge about the environment. Range queries are initiated to meet the requirement. The format of a range query is written as *<Attribute ID, Attribute value range, Timestamp>*. Using the object model defined in Figure 2(B), we can write a range query as: *Query (101, [0, 3], 300)*. This expression means that an ALP is requesting the *X-pos* value of all entities (tiles and holes) located at left side of *X-pos* 3 at time 300. The ALP will be returned with a set of SSVs in format: *[<SSV ID1, value 1>, …, <SSV IDn, value n>]*. The common attributes are useful in this case: an ALP can retrieve whatever SSVs blindly regardless of what sub-type those SSVs belong to.

In contrast, an SSV Query accesses an individual data item (SSV) in the environment which have already been known to the agent. The SSV query targets on a single SSV according to its specified ID. Thus, SSV query is also called ID query, including SSV Read and SSV Update. The SSV read has a simpler format: *<SSV ID, Timestamp>*, and a successful read will return a single SSV: <SSV IDx, Value x >. For example, an SSV read, *Read (3000101, 500)*, means an ALP needs to know the value of tile3000's *X-pos* at time 500 (see Figure 4). An SSV Update writes new value and the associated timestamp to a designated SSV, for example: *Update (3000101, 7, 501)*.

Only when an ALP needs, can an object (and SSVs associated with it) be created and deleted. Creation and deletion of SSVs are performed through CLPs eventually. SSVs can also be moved, whether moving an SSV and to where should it be moved totally depend on the load management mechanism [20].

[2] The scheme is similar to the RTI "handles" defined in the HLA specification. The integer representations remain unchanged within a simulation session and are unique in all ALPs and CLPs. From the value of such an integer, we can derive the entity/abstracts it represents. For example, an attribute "101" means the attribute "*X-pos*" of "*Tile*". Similarly, the SSV ID refers to an attribute (or a set of attributes) of a unique object.

B. Dynamic Distribution of Shared State Variables

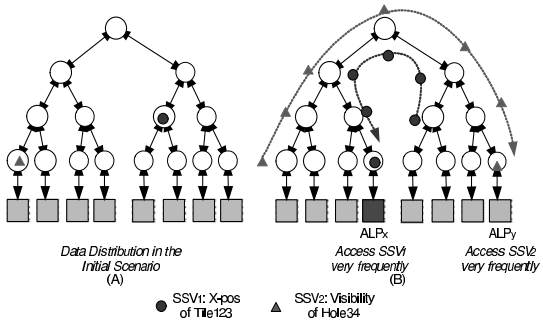

Fig. 5. Moving Shared State Variable for Load Management

The hierarchical infrastructure clusters ALPs and shared state according to the ALPs' *SoI*. As discussed in [20], the DS-MAS framework enables load management via approximating the ideal *SoI*. In this project, the term load management is slightly different from that defined in the context of conventional distributed systems, which does not simply mean migrating load from congested host to another idle one to balance the utilization of system resource. In DS-MAS, load management focuses on optimisation of the cost of managing shared state according to the statistics of data accessing. The "load" includes the cost of the communication and computation for accessing and managing shared state. The definition of access cost and the SSV migration algorithm are detailed in [20]. As a consequence of load management, the CLPs will gradually move the corresponding SSVs closer to those client ALPs which access them the most frequently. Figure 5 illustrates this SSV migration procedure and its ultimate effect.

Figure 5(A) gives the initial scenario of a DS-MAS simulation, in which shared state variable SSV_1 ("*X-pos*" of the object "*Tile123*") and SSV_2 ("*Visibility*" of the object "*Hole34*") are distributed as in the figure. At some stage, ALP_x is interested in SSV_1 and starts to access SSV_1 frequently, so does ALP_y for SSV_2. Here we assume other ALPs access these two SSVs rarely comparing to ALP_x and ALP_y. As controlled by the load management algorithm, SSV_1 and SSV_2 should be gradually moved in the tree towards ALP_x's server CLP and ALP_y's server CLP respectively. With this procedure continues, eventually the two SSVs will be redistributed as in Figure 5(B).

C. Tradeoffs in Defining Shared State Variables

An SSV may represent an attribute or a set of attributes of an object. It is simulation modellers' responsibility to decide how to define SSVs, whether assigning a simple type or a composite type to it. For example, a modeller can define the *X-position* or *Y-position* of tiles as two different simple SSV types. Alternatively, the modeller may choose a combination of the two attributes <*X-pos*, *Y-pos*> of tiles as a composite SSV type. The two different schemes of modelling SSVs make significant difference in distributing SSVs.

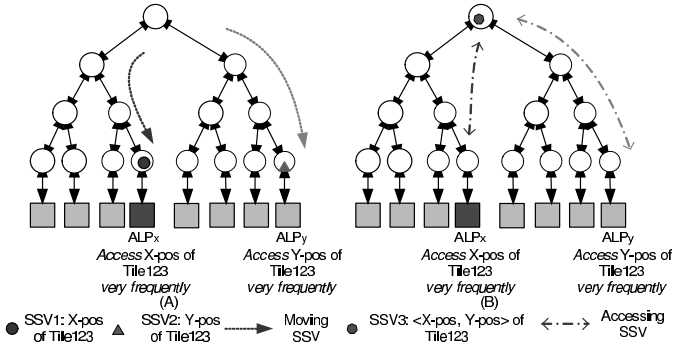

Fig. 6. Data Re-distribution for Defining SSVs in Single Type and Composite Type

As illustrated in Figure 6, ALP_x and ALP_y are interested in the attributes *X-pos* and *Y-pos* of object *Tile123* respectively. If SSVs are defined as single types, there will be two individual SSVs (SSV_1 and SSV_2) representing the two attributes of *Tile123* (Figure 6(A)). With the advancing of simulation, SSV_1 and SSV_2 will be moved forward to the server CLPs of ALP_x and ALP_y. If a composite type <*X-pos*, *Y-pos*> is used to denote the two attributes of tile objects, there will be a single SSV for the *X-pos* and *Y-pos* of *Tile123*. In this case, ALP_x and ALP_y are interested in different portions of the same SSV (SSV_3). Suppose that the two LPs access SSV_3 in similar frequencies, SSV_3 will be moved to a CLP at the "middle point" between them.

Modelling SSVs with what type also affects data accessing in both design complexity and system performance. ALPs may issue queries with a set of attribute value ranges. For example, an ALP requests the information of the tiles within the area ($4 \le X\text{-}pos \le 6$) AND ($5 \le Y\text{-}pos \le 7$). When adopting the simple SSV type scheme, the server CLP has to search all the tile objects meeting the conditions on *X-pos* then on *Y-pos*. Subsequently, the ALP will be returned with the intersection of the results obtained in the two searches. In this case a lot of redundant data are accessed and transmitted unnecessarily. If SSVs are defined using the composite type <*X-pos*, *Y-pos*>, the SSVs meeting both conditions can be identified in a single search. Comparing to the simple type scheme, the composite type scheme can significantly reduce the cost of searching multiple attributes of the same object type.

Undoubtedly, defining SSVs in different combinations of the same set of attributes will also have impact on data management. Both SSV modelling schemes have their advantages and drawbacks, and we leave the options to the modellers.

IV. Data Accessing in DS-MAS Based Distributed Simulations

Another key issue in data management is to provide efficient data accessing. The first crucial task of any operation on SSVs is to locate the target(s). Routing solutions are needed for ALPs/CLPs to locate (1) SSVs according to the attribute value ranges (range query) and/or (2) a particular SSV from the given ID (other operations including ID query and update). The two

types of locating significantly differ from each other. The former one searches for a group of SSVs based on the specified common attributes and constraints, while the targets are versatile and dynamic in each query. This task is similar to multicast on dynamic group. In contrast, the latter one accesses an SSV from its ID. It is specific and static, basically a unicast on mobile destination.

In order to minimise the computational and communicational overhead, which is also the goal for load management, following principles should be conformed to in choosing the routing solution:

1) To avoid unnecessary data transmission and complex searching as much as possible.
2) To minimize the number of times of initiating global communication or computation in performing frequent executions.

A. Assumptions

SSV queries and updates are the most basic operations issued by the ALPs. SSVs can be moved between CLPs on the initiative of load management, and such move is controlled by the load management mechanism according to the statistics of a large number of SSV queries/updates. ALPs may create objects and delete them when necessary, and these operations are once off for each object (or SSV) and can be treated as transient operations. Thus in general, the frequencies of these operations on SSVs follow the order: *Query/Update >> SSV Move, Object Creation/Deletion*.

For query on SSVs, the data being transmitted include (1) searching criteria information contained in a query, (2) update/returned value of an SSV or (3) returned values of a set of SSVs. It is likely that the payload sizes of these data items comply with the sequence: *Values of SSV set >> Update/Value of an SSV, Attribute value range Query > SSV ID Query*.

Locating an SSV includes two element executions: (1) forwarding the request amongst CLPs and (2) looking up the SSV inside a CLP. The inter-CLPs transmission overhead is significantly higher than the overhead of performing simple local computation: such as searching within a finite list. It is hard to reduce the transmission overhead with existing networking technologies, which increases linearly with the number of hops to be traversed. On the contrary, local computation is relatively easier to be optimized, for example using advanced algorithms or high performance machines.

Derived from above assumptions, the following specific factors should be treated with high priority when making tradeoffs in designing a routing algorithm:

1) Optimization of the query/update procedures.
2) Minimisation of the hops to be traversed in frequent data transmissions.
3) Preference on locally searching information for SSVs rather than resort to remote nodes.
4) Reduction of the overhead in transmitting the values of SSV sets.

Undoubtedly, any routing solution should guarantee that it will never miss any target in the system meeting the query condition. All efforts to explore accurate routing solutions/reduce redundancy of routing should be based on the premise of that.

B. Routing in the Communication Logical Process Tree

Routing accesses to SSVs can be performed via either their location information in the CLP tree or the attribute value range information maintained at the ports of the CLPs [16]. SSVs may be moved between CLPs, but there are no multiple copies of a single SSV existing in the overall system. The status of an SSV can be altered due to updating and load management (except object deletion). Update may change the value of the SSV which directly affects the corresponding attribute value range. Load management may induce the migration of the SSV around the CLP tree. Thus, the location of this SSV in the system changes and so does the value ranges at related ports. This immediately influences the ID query on this SSV and possibly the range query.

To ease locating SSVs in the CLP tree, it is necessary to code[3] the tree to identify the CLPs. The address of an SSV is defined as the code of CLP at which it is maintained, the CLP is named as the host CLP of this SSV, and this SSV is called a local SSV of this CLP. When forwarding data, a CLP can decide from which port to push the access request to the destination CLP.

The fixed architecture of a CLP trees determines that: (1) an SSV can only be moved from a CLP to its direct neighbours, and (2) between any ALP and CLP, there exists only one exclusive path. Once the target SSVs are located, the returned data need to be simply conveyed to along the path (the query just traversed) in a reverse direction to the source ALP (always linking to the leaf CLP nodes).

C. Address-based Routing

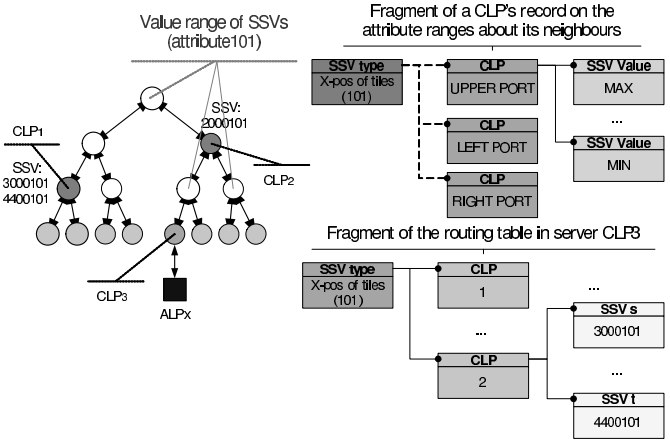

Fig. 7. Illustrating a Scheme for Address-based Routing

Address-based routing searches for an SSV (or SSVs) according to its (their) addresses. Figure 8 illustrates an address-based routing scheme, which binds the ID of an SSV to its address. Each server CLP maintains a routing table containing the addresses of SSVs that have been accessed by any client ALP. The routing table has a hierarchical format

[3] We do not prefer any particular coding scheme, as long as the code it specifies to each CLP is unique and remains consistent in the simulation.

using attribute IDs as indexes. From a particular object attribute's perspective, the table maintains the addresses of CLPs hosting the same type of SSV. The SSV IDs of this type are recorded under the host CLP entry. The direct neighbours of a CLP keep the attribute value ranges of SSVs on the CLP. Each port of a CLP also stores the attribute value range information on the direct neighbour CLP beyond that port. This information is obtained and refreshed when the updates on those SSVs occur. For example in Figure 7, there are several SSVs representing the *X-pos* of tile objects in the CLP tree, and two of them are maintained by CLP_1.

1) Range Query with Address-based Routing

The information in a range query consists of two parts, i.e. the attribute ID(s) and the query window on attribute value. When an LP issues a range query, the server CLP is responsible for routing the query to the destinations. Figure 8 depicts the routing algorithm. First, the attribute ID will be used to check through the routing table (see Figure 7). In the case that the SSV type (see Section III) is absent in the table, the server CLP has to initiate a global search on other server CLPs to collect information about the SSVs of the queried type, such as SSV IDs and their addresses. Subsequently, the server CLP may establish a new entry about the SSV type.

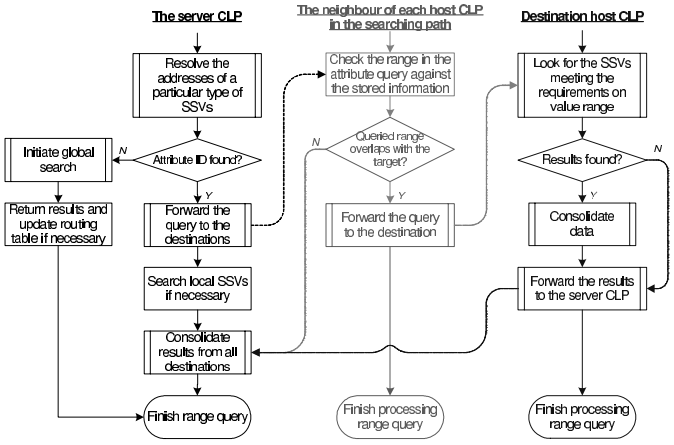

Fig. 8. Address-based Routing for Range Query

If the entry is found, the addresses of the host CLPs can be resolved and the server CLP "multicast" the query to the destinations. Prior to entering into each individual host CLP, the algorithm needs to check the attribute value range information maintained in the corresponding port of the neighbour CLPs. Only when the query window overlaps with the stored attribute value range will this host CLP be searched. The results will be returned to the server CLP and passed to the client LP eventually.

2) Shared State Variable (ID) Query

The algorithm for an ID query is straightforward. When an ALP issues an ID query, the server CLP locates the destination (if the SSV is not maintained locally) from its routing table and forward the query to the particular host CLP. After which, the value will be fetched from the host CLP.

The server CLP locates an SSV prior to updating its value in the same way as routing an ID query. When the value of the SSV is updated, the host CLP computes the attribute value range. If the range changes, the updated range information must be and only be forwarded to the direct neighbour CLPs.

3) Migration of Shared State Variable

When an SSV migrates, the change of SSV's address immediately affects the ID-to-address binding. The routing tables on the server CLPs have to be updated with its new address. Immediate broadcasting the new address with each move of any SSV will be too costly considering the number of SSVs and times they may be moved. Instead, we propose a gradual address updating approach to avoid any global propagation for updating addresses.

As shown in Figure 9, the approach makes use of the fixed architecture of the CLP tree. The port from which an SSV is moved is recorded in the original host CLP, and the CLP becomes the SSV's *correspondence CLP*. The map between port and migrated SSVs is looked up as another routing table for searching those SSVs. As illustrated in Figure 10, when ALP_x accessed SSV_1 earlier, CLP_m was SSV_1's original host CLP. Obviously, at that time the query from ALP_x on SSV_1 must be routed to CLP_m's right port along a fixed path (namely *original path*, see Figure 9).

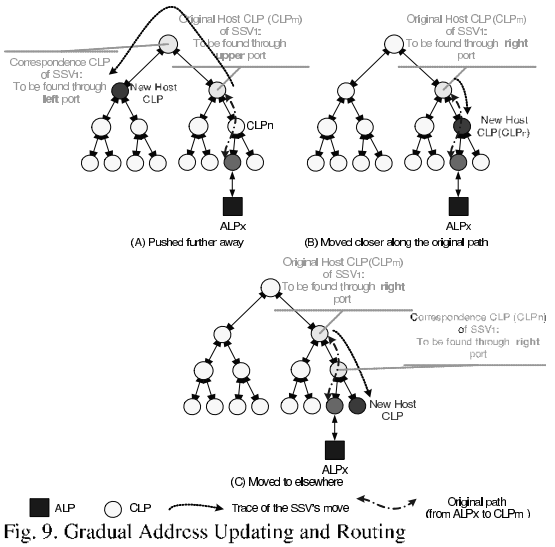

Fig. 9. Gradual Address Updating and Routing

After SSV_1 migrates to a new host, from ALP_x's point of view, there are three different cases: (1) SSV_1 has been pushed further away (Figure 10(A)), (2) SSV_1 has been brought closer to ALP_x (Figure 10(B)) or (3) SSV_1 has been moved to elsewhere (Figure 10(C)). For case (1), when a new query reaches CLP_m, it will be forwarded to CLP_m's direct neighbour beyond its upper port[4]. The forwarding will be relayed until the new host is found. For case (2), the new host must be an intermediate node in the original path, and the query will be answered straightforwardly when reaching the host. For case (3), along the original path, the query will pass a correspondence CLP_n, and then it will be relayed downwards to the new host CLP of SSV_1. From the above discussion, no matter in which new host an SSV locates,

[4] Case (1) also applies when SSV_1 has been moved out from CLP_m's left port.

the query will only travel along the exclusive path between the ALP and the new host. The routing procedure does not concern any CLP which locates off the path. The routing table of the server CLP will be updated with the returned information. As soon as all related server CLPs (whose client LPs have accessed the SSV before the SSV being moved) have retrieved the new address, the correspondence CLPs can simply release the record about this SSV.

As a result of migrating an SSV, the value range of its type may change in both the original and the new host. Therefore, the attribute value ranges related to the two hosts have to be re-computed and updated. However, this does not influence the routing on other SSVs directly.

D. Range-based Routing

As introduced in Section II, the ports of CLPs are specially designed to maintain the attribute value ranges, which are located beyond each port. Using the attribute value range information of SSVs can provide another candidate routing solution, namely range-based routing. Under this solution, a CLP forwards a query according to (1) the availability of the SSV type being queried beyond its ports and (2) the value range of SSV(s) belonging to the given type.

1) Range Query with Range-based Routing

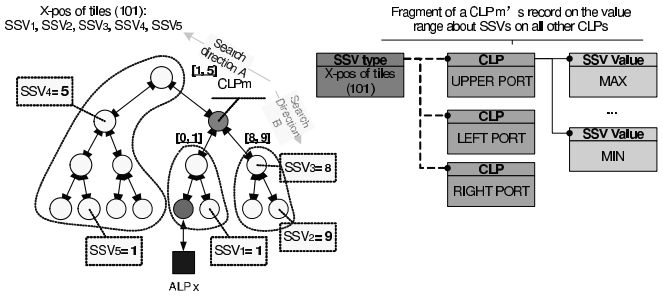

Fig. 10. Illustrating a Scheme for Range-based Routing

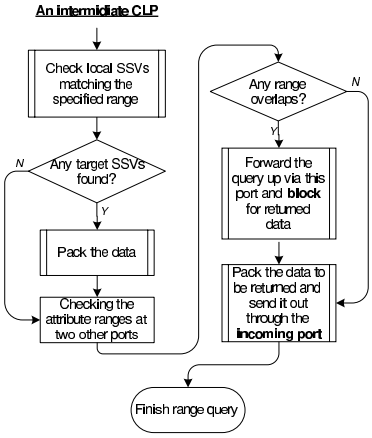

Fig. 11. Range-based Routing for Range Query

The range-based routing matches the query window with the attribute value ranges along the searching paths to gradually approaching the potential targets. Searching will be given up at the directions where the query window and attribute value ranges do not overlap. In the example shown in Figure 10, CLP_m keeps a record of the SSVs within the circles in its three ports respectively. Suppose that a range query, *Query (X-pos, [2, 6], 300)*, reaches CLP_m, it has two optional directions to relay the query. Direction *B* will be given up as the attribute value range *[8, 9]* does not overlap the window *[2, 6]*. The query will be only forwarded through direction *A* as attribute value range *([1, 5])* for this direction matches the condition. When an ALP issues an attribute value range query, a range-based routing will start from its server CLP. A range-based routing algorithm searches the desired SSVs in following steps, used by *any* CLP which receives a request (Figure 11):

1) The CLP checks the values of local SSVs against the query window, packing the target SSVs matching the conditions (if any).
2) The CLP computes the attribute value ranges maintained by the two other ports (rather than the incoming port, i.e. the port accepts this query), after which forwarding the query through the port with overlapped range (if any).
3) If neither attribute value ranges matches the query, the CLP will return the local target SSVs or "no-result" message to the incoming port.
4) If either attribute value range matches the query, the CLP blocks for SSVs to be found or "no-result" message from the outgoing port and returns the results to the incoming port.
5) The searching for current query via this CLP terminates, the control is passed to the direct neighbour CLP beyond the incoming port.

2) Influence of Definition of Shared State Variable on Ranged-based Routing

Using simple SSV type has particular influence on the range-based routing when querying with multiple attributes. Figure 12 presents a snapshot of simulation in which an range query (see Section III.C) is under processing. CLP_2 maintains the *X-pos* of *Tile123* and the *Y-pos* of *Tile456* while CLP_3 maintains the *Y-pos* of *Tile123* and the *X-pos* of *Tile456*. When the query is forwarded to CLP_1 and CLP_1 needs to decide the direction for routing. It is obvious that *Tile123*: <5, 6> meets the conditions. However, if only using the combined range information as criteria, the search will stop at direction *A* since the range of *Y-pos* (*Tile456.Y-pos* = 2) does not match, and it is the same for direction *B*. Hence, ALP_x will get no result although the target SSVs do exist. This problem is due to that the SSVs associated with the same object are distributed arbitrarily and the query conditions are set based on attributes rather than ID or any SSV-specific information. In this case, the routing algorithm has to search each individual attribute in an atomic range query and consolidates the results afterwards.

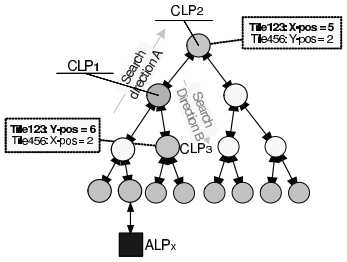

Fig. 12. Problem with Range-based Routing using Simple SSV Type

3) Segmenting Attribute Value Range

As discussed previously, a CLP decides whether to forward a query through a port totally relying on attribute value ranges beyond that port. The range-based routing algorithm is sensitive to SSV updates, by its nature. When an SSV update occurs, the value range of its attribute may change in the host CLP. In order to ensure accurate routing, the changed range has to be sent to the CLPs being affected by checking against the corresponding range. This global computation is expensive as it may occur for each update on any SSV, which explicitly conflicts with the principles of designing routing algorithms (see Section IV.A). In order to reduce the overhead, a value range segmenting approach is proposed.

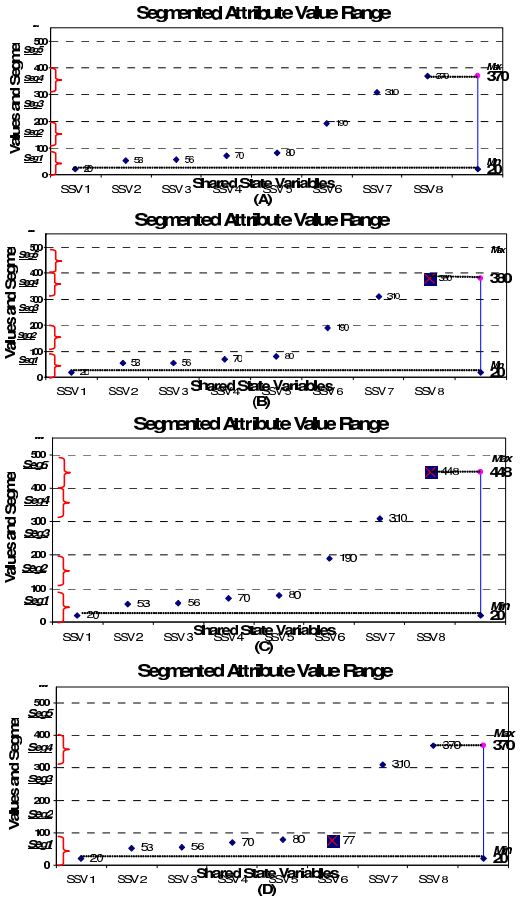

Fig. 13. Segmenting Attribute Value Range

For example (see Figure 13(A)), a CLP contains a set of SSVs (*X-pos* of tiles) with values listed as: *{20, 53, 56, 70, 80, 190, 310, 370}*. Instead of using a simple range description like *[Min(20), Max(370)]*, we segment the value space, such as *{Seg1: [0, 100], Seg2: [100, 200], Seg3: [200, 300], Seg4: [300, 400], Seg5: [400, 500], …}*. The approach logs the number of SSVs whose values fall onto each segment. In Figure 13(A), we have *{Seg1(5), Seg2(1), Seg3(0), Seg4(2), Seg5~n(0)}*. The attribute range is written as *{Seg1∪Seg2∪Seg4}*. Those segments containing no SSV are excluded from the expression. Therefore, the attribute range does not need to be updated continuously, with the frequency proportional to the granularity of segments. Obviously, the segmented range should always cover the values of all SSVs it represents.

When an update occurs, even if the value of the updated SSV is beyond the original *[Min, Max]*, the approach may not require altering the segmented range immediately. For example, suppose the SSV with value 370 is updated to 380, the updated value still falls onto *Seg4* (Figure 13(B)). The range has to be updated only when there is no SSV having a value in one of the segments (narrowing, see Figure 13(C)) or any SSV's value exceeds all existing segments (expanding, see Figure 13(D)). However, the algorithm designers have to make tradeoffs between the accuracy of routing and the efficiency of computing ranges (see Section V), both subject to the granularity of segments.

4) Segmenting the IDs of Shared State Variables

Each SSV is given a unique integer ID (see Section III), for example two SSVs have ID 3000101 and 4400101 in Figure 8. Similar to using attribute value ranges, we can calculate the ranges of these integers (ID range) and record those at the ports of the CLPs. Thus, ID Query can be routed based on the range of IDs.

Load management has direct influence on the ID range. After an SSV migrates, the ID range of its original and new host CLP may change. This issue is similar to dealing with the SSV update in attribute value range query. The segmenting approach also applies in manipulating ID queries.

5) Comparison between Routing Solutions

The two categories of routing solutions involve different research issues and problems. To make trade-offs amongst these issues is a challenging work. In this section, we make a qualitative comparison showing the advantages and disadvantages of both solutions. The comparison is detailed as follows referring to the algorithms described in Section IV.C&D.

a) Availability of Shared State Variables

Given an SSV type of any ALP's interest, both solutions provide correct information of where those SSVs may be available. The two solutions avoid routing accesses to those CLPs which do not contain any SSV of the requested type. The major difference is that the address-based solution gives exact locations while the range-based solution directs the accesses to the correct searching paths. It is likely that the range-based solution can provide more accurate routing for range queries than address-based routing, and vice versa for ID queries.

b) Efficiency in Performing Range Queries

When performing range query, the range-based solution forwards the query to the potential targets in a bottom-to-top manner. In the target narrowing procedure, those out-of-range SSVs can be filtered out effectively under some particular situations, e.g. the SSVs locate in the CLPs dispersedly and their values distribute sparsely in the value space. The address-based solution needs to route the access to the neighbours (in the searching path) of all CLPs to match the

ranges between the query's and the host CLPs'. In general, the ranged-based solution forwards range query to a smaller set of CLPs than the address-based solution does.

The two solutions require searching within the same number of potential host CLPs. The solutions do not make any difference in the complexity of searching within those hosts or the overhead in delivering results to the requesters (some ALPs).

c) Complexity for Maintaining Range Information

As discussed in section IV.D, the range-based solution intensively relies on the attribute value range information. From a CLP's point of view, the attribute value ranges (on the other CLPs beyond each of its ports) must be available and accurate. A ranged-based algorithm needs to manage the segmented ranges properly. Setting precise segment can obtain accurate routing while it tends to result in a wide range update amongst other CLPs with high frequency. Adopting coarse segment will reduce broadcasting of updated ranges but accuracy of using attribute for routing will be impaired.

It is unnecessary for the address-based solution to maintain and broadcast range information, by this solution's nature. A CLP only needs to simply compute its local attribute value range (*[Min, Max]*) prior to notifying its direct neighbours.

In the case of handling SSV migration, the address-based solution does not incur any extra communication overhead for routing. In contrast, the range-based solution has to concern the immediate impact on the previous and current host CLPs, and it may involve updating attribute value ranges on the CLPs.

d) Efficiency in Shared State Variable Query

The address-based solution is able to resolve the address at the server CLP immediately for an SSV ID query, under which an SSV can be accessed via a fixed path without routing to irrelevant CLPs. Using the range-based solution, querying an individual SSV is not straightforward.

e) Complexity for Maintaining Routing Information

The address-based solution assigns different tasks to server CLPs and other CLPs. A server CLP keeps addresses of all SSVs of its client ALPs' interest. However, address resolving within a centralized node can be optimised. The address change of any SSV does not affect routing. The range-based solution distributes the routing information all over the CLP tree in an implicit manner. The address changing of any SSV may affect multiple CLPs or even the whole CLP tree.

f) Design Complexity

The address-based solution assigns different tasks to server CLPs and other CLPs. A server CLP keeps addresses of all SSVs of its client ALPs' interest. However, address resolving within a centralized node can be optimised. The address change of any SSV does not affect routing. The range-based solution distributes the routing information all over the CLP tree in an implicit manner. The address changing of any SSV may affect multiple CLPs or even the whole CLP tree.

V. Experiments and Results

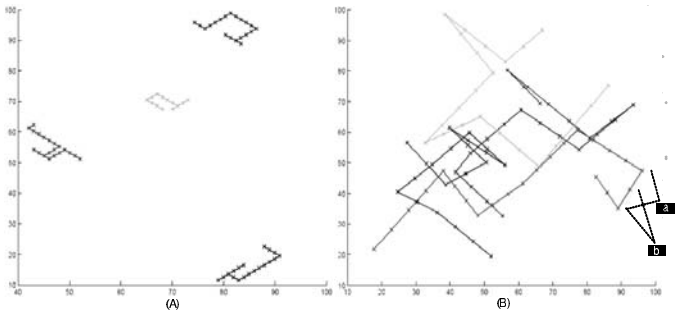

Fig. 14. Illustrating Different Agent Movement Patterns

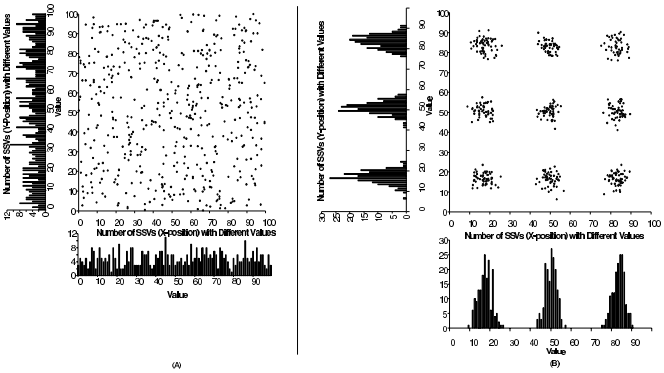

Fig. 15. Illustrating Different SSV Value Distribution Patterns

A quantitative comparison of the two proposed solutions is not trivial, as it involves the evaluation of a number of complicated factors. Those include (1) distribution pattern of the value of SSVs in the value space (*SSV Value Distribution*), (2) the behaviour or access pattern of ALPs on SSVs, (3) the physical distribution of SSVs on the whole CLP tree (*SSV Distribution Pattern*), and (4) the distribution of the value of SSVs in each individual CLP[5]. Factor 1 and 2 are at application level related only to the agents and their environment whereas factor 3 and 4 are at simulation level and can be controlled by users[U5]. Moreover, some factors have particular influence on the range-based routing solution, and those are (5) the *ratio* of number of updates to number of range queries, (6) the *fluctuation*s between the updated value to the original value, and (7) the *granularity of value segment*s.

From the scale of CLP tree and number of SSVs, it is relatively straightforward to estimate the computational and communicational complexity of the address-based routing solution using mathematical approaches. However, evaluation of range-based routing needs to consider other complicated factors at both application level and simulation level. For the sake of a quantitative analysis, one approach would be to directly implement and integrate the two solutions into the DS-MAS kernel. However, this would require considerable implementation efforts, and at least part of the implementation

[5] This is different to the aforementioned value distribution parameter. For example, in a scenario the value ranges of SSVs in all CLPs are very close, and in another scenario the value ranges are significantly distinct from one CLP to another.

could be in vain, as the strategies may not meet the performance requirements. To avoid this, and to provide a generic framework for the study of dynamic data access in distributed systems in general, we have adopted a meta-simulation approach, as proposed for instance in [13][14][22][24].

A. Performance Evaluation Meta-simulation Model

The design of the performance evaluation meta-simulation follows a layered approach similar to [9]. The application model is built at the top layer and responsible for generating realistic query patterns. The next layer is the middleware layer, where the routing strategies are described and the DS-MAS framework is represented. The third layer, which typically is reserved for the network model, is implicitly represented in the performance measurements which are integrated by calculating the costs of queries in the second layer. Thus, similar to many simulations of P2P systems, the characteristics of the underlying network are abstracted away by only counting hops and messages.

The application model focuses on the simulation of situated agents, namely, an agent has a position that determines its region of interest: only objects situated in the region can be accessed by the agent. In addition, situated agents are usually able to change their own positions. This behaviour was modelled for a two dimensional environment, as shown in Figure 14. An agent moves step-wise towards a pre-selected target along the shortest path, and it randomly chooses a new target on arrival. The distance an agent can move in each step is written as *step size* (mark "a"), which reflects the rate of change of the agent's access pattern. The distance of the new target, the target distance (mark "b"), is defined by the number of steps it takes the agent to reach it. The step size and target distance determine the activity scope and movement speed of an agent. After each step of movement, an agent generates ID or Range queries concerning its actual region of interest. Figure 14 plots the traces of agents' movement with step size = 1 (A) and step size = 5 (B).

All SSV types within the MAS model have a spatial meaning, i.e. the value ranges for Range queries reflect the actual positions of the agents. Each SSV type represents a certain dimension of the environment, such as "*X-Pos*" or "*Y-Pos*". SSVs may have a uniform value distribution or multiple normal distributions, and those are illustrated in Figure 15(A) and (B) respectively.

The model of the DS-MAS framework is formed by a set of SSVs. Each SSV consists of (unique) ID, type, value and position in the CLP tree. The modelled CLP tree is binary and complete and therefore its structure can be defined by its depth. Another important parameter is the number of segments used by both routing algorithms, which determines the granularity of the description of the value distribution of SSVs. To eliminate a possible source of bias, no load management mechanism has been modelled, i.e. the SSVs could not migrate (as proposed in [20]), although the SSVs' distribution pattern differs for different runs. Nevertheless, the mutual impact of routing and load management could be considerable and should be subject of future research. The model was simulated using discrete time steps.

B. Setup of Experiments

The environment for the experiments is set as a 2-dimensional space containing 6,200 objects of 8 types. The experiment model defines SSVs as the *X-pos* or *Y-pos* of any object in the environment (12,400 SSVs of 16 types) and allows SSVs having numerous value distributions. On initialisation, there are 64 agents locating at any position in the space with identical probability. The step size per dimension and the target distance conform to normal distributions ($\mu = 2.0$, $\sigma = 1.0$) and ($\mu = 5.0$ and $\sigma = 2.0$). Each agent executes 300 time steps, and per time step it generates 8 requests for any type of SSV randomly. The diameter of an agent's region of interest is set to 2.0. The default parameters for the experiments are summarised in Table 2 and 3.

The physical distribution of SSVs can be alternated by two parameters, namely *root imbalance* and *CLP fluctuation*. Root imbalance describes the percentage of additional SSVs hosted by the root CLP comparing to the rest. When root imbalance = 0, SSVs are distributed evenly on all CLPs. When root imbalance = 1, all SSVs are concentrated on the root CLP. The CLP fluctuation constraints the maximum difference between the greatest and smallest value of SSVs of the same type on an individual CLP. For example, suppose CLP fluctuation = 0.05 and the designated type of SSVs can have a value space from 0 to 100, then the difference of these SSVs' value on the CLP is not greater than 0.05× 100 = 5: A CLP may host two SSVs of type X with values 80 and 82, but another SSV of type X, with value 75, cannot be hosted by the same CLP, because 82−75 > 5. Since SSVs are not moved but have dynamic values, this condition holds only for the initial state.

The experimental results are reported in terms of routing cost and accuracy. The routing cost is measured using two metrics: the *number of messages* and the *number of hops to be traversed* in resolving each access on SSVs. The number of messages is the number of all messages that are generated by the routing algorithm in order to resolve a query. Hence, the number of messages is a measurement of the overall bandwidth consumption. The number of hops is the maximum number of messages that had to be sent sequentially until the request could be resolved. This means, that the number of hops corresponds to the maximal path length from the ALP generating the request to a CLP which had to be contacted, multiplied by 2 (for the query and the corresponding response). Hence, the number of messages is a measurement of the overall latency. The two metrics for the routing algorithms are denoted by variables $range\text{-}based_{messages}$, $range\text{-}based_{hops}$ (for range-based algorithm) and $address\text{-}based_{messages}$, $address\text{-}based_{hops}$ (for address-based algorithm).

In order to calculate the accuracy of the routing algorithms, we compute the minimal number of messages and hops ($opt_{messages}$ and opt_{hops}) which is the absolute limit for optimising the cost of any routing algorithm. Figure 16 gives an example of querying two SSVs with the smallest set of CLPs and connections for this query highlighted. Each connection needs

to transmit two messages (one request and one response), thus the total number of messages is 10. The maxim number of hops is record as 8 (for reading the SSV on the left), this is because messages have to be sent sequentially (in any path) until reaching the target and the latency in other concurrent search paths are masked. The overhead between the ALP and its server CLP are not counted. The ratio of the minimal number of messages (or hops) to the number of messages (or hops) is computed as the *accuracy* of routing algorithms. For example, the "message accuracy" of ranged-based algorithm is written as $accuracy_{messages}^{range-based} = \frac{opt_{messages}}{range-based_{messages}}$. Likewise, we can calculate the "hop accuracy" for ranged-based algorithm ($accuracy_{hops}^{range-based}$) and accuracy of address-based algorithm ($accuracy_{messages}^{address-based}$ and $accuracy_{hops}^{address-based}$).

Scalability is also an important issue. Nevertheless, for the sake of simplifying the experiment design and focusing on the factors listed above, we choose to fix the depth of the CLP tree, the number of ALPs and SSVs in most experiments. The outputs of the experiments will be checked against the qualitative analysis on the algorithms in Section IV. Hence, the system designers can decide what kind of solution to be adopted in designing DS-MAS framework like systems.

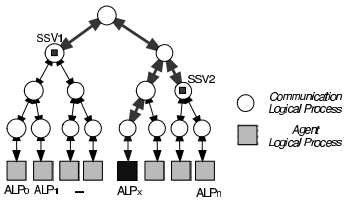

Fig. 16. Minimal Number of Messages and Hops (Optimal Cost) for Routing

TABLE II
SUMMARY OF THE DEFAULT APPLICATION LEVEL PARAMETERS

Name of Parameters	Value
Time steps	300
Number of agents	64
Number of dimensions of the environment	2
Value space at each dimension	[0, 100)
Number of SSV types	16
Shared State Variables	12,400
Value space for each SSV type	[0, 100)
Events to be generated per agent per time step	8
Step size of all agents per dimension	μ = 2.0, □= 1.0
Target distance of all agents in steps	μ = 5.0, □= 2.0
Agent's range of interest	2.0

TABLE III
SUMMARY OF THE DEFAULT SIMULATION LEVEL PARAMETERS

Name of Parameters	Value
Depth of the CLP Tree	4
Number of client ALPs to each server CLP	4 ALPs per server CLP
Root Imbalance	0
CLP Fluctuation	1
Number of segments for routing algorithms	100

C. *Effect of Agent's Environment (SSV Value Distribution)*

To test the effect of the features of multi-agent systems, we adopt numerous non-uniform distributions of SSV values while using default values for all other parameters. Values have been assigned to SSVs in a round-robin manner by one of three normal distributions (see Figure 16(B)). The mean values of normal distributions are $16\frac{2}{3}$, 50 and $83\frac{1}{3}$. The deviation □ is varied from 0 to 10 to make SSV value distribution change from highly concentrated to highly scattered. Figure 17 reports the effect of value distribution on the cost of routing algorithms vs. the optimal cost.

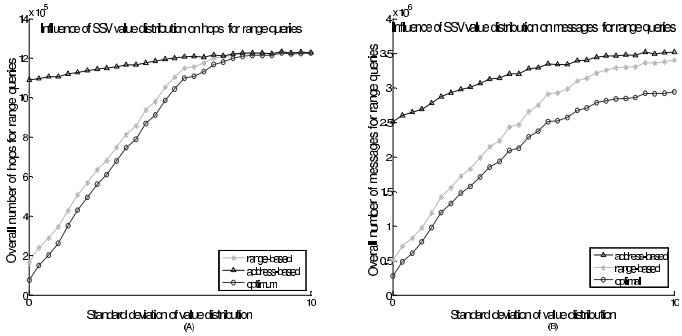

Fig. 17. Influence of Overall SSV Value Distribution on Routing Cost

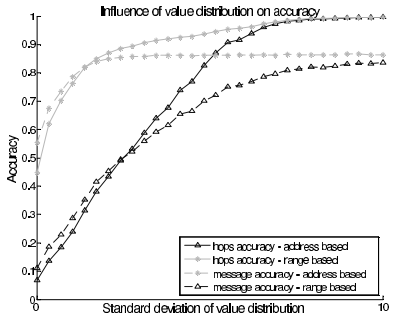

Fig. 18. Influence of Overall SSV Value Distribution on Accuracy of Routing

The range-based algorithm performs much better in terms of number of hops, by closely approaching opt_{hops}. However, given that values of SSVs are scattered enough, the number of hops by both algorithms converges to opt_{hops}. With values of SSVs distributed more sparsely, routing becomes more costly and neither algorithm approaches $opt_{nessages}$. The effect of SSV value distribution is further emphasized in Figure 18. The $accuracy_{hops}$ of both algorithms converge to 1 while the maxim of $accuracy_{messages}$ for both algorithms only approximates 0.8. In all situations, the range-based algorithm is likely to incur less overhead for routing range queries.

Experiments have also been performed to measure the accuracy of making ID queries. The address-based algorithm always achieves optimal results (accuracy = 1) while the accuracy of the range-based algorithm is quite low, about 0.36. This is due to that the range-based algorithm does not store the location of an individual SSV and requires searches using range-based techniques on SSV IDs (Section IV.D.4).

D. *Effect of SSV Value Distribution Pattern*

A set of experiments have been performed to examine how the distribution pattern of SSVs in the simulation infrastructure (the CLP tree) affects the performance of routing algorithms. This subsection reports the effect of the simulation level factors (1) the physical distribution of SSVs on the CLP tree, and (2)

the distribution of SSVs' values on each individual CLP, given the behaviour of agents remains the same in these experiments. Two parameters, *root imbalance* and *CLP fluctuation*, are varied between 0..1 and 0.05..1 respectively. Figure 19 gives the accuracy of range-based algorithm for ID query in terms of messages. The SSV value distribution on a CLP does not affect the routing of ID query at all. However, concentration of SSVs on fewer CLPs will dramatically improve the accuracy of routing ID queries using range-based algorithm.

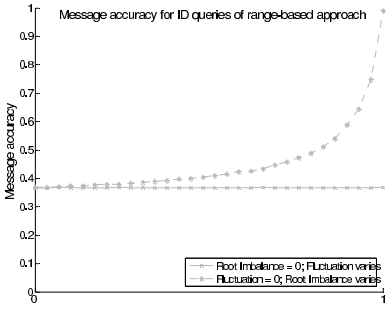

Fig. 19. Message Accuracy for ID Query

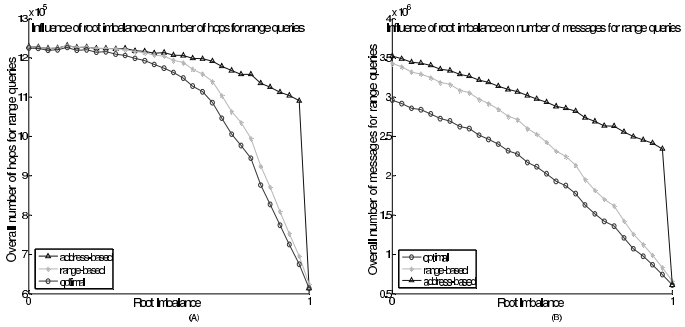

Fig. 20. Influence of the SSV Distribution Pattern on the Routing Cost

Figure 20 illustrates the impact of the SSVs' physical distribution pattern on the cost of routing range queries using different routing algorithms. When SSVs are distributed on all CLPs uniformly (root imbalance = 0), $range\text{-}based_{hops}$ and $address\text{-}based_{hops}$ are nearly equal to opt_{hops} while $range\text{-}based_{messages}$ and $address\text{-}based_{messages}$ are very close but still much greater than $opt_{messages}$. With the increase of root imbalance, the routing cost gradually decreases. The range-based algorithm adapts much better than the address-based algorithm. When root imbalance = 1, all SSVs locate at the root CLP and makes no difference to either algorithm. However, this extreme case only reflects the idea of using centralised CLP (see Section II).

Figure 21 presents the cost of performing range queries against the value distribution on each CLP. The results are similar to those obtained in Figure 17, which means the value distribution of all SSVs and the value distribution of SSVs on each individual CLP both have significant influence on the routing cost involved in range queries.

The accuracy of routing algorithms against the SSVs' physical distribution pattern is illustrated in Figure 22. The results further indicate that the range-based algorithm adapts better to the SSV's distribution pattern. However, the latencies (number of hops) incurred by both algorithms are similar while the range-based algorithm generates much less communication traffic (number of messages). Furthermore, comparing to Figure 18, the value distribution on an individual CLP makes less significant impact than the overall value distribution does.

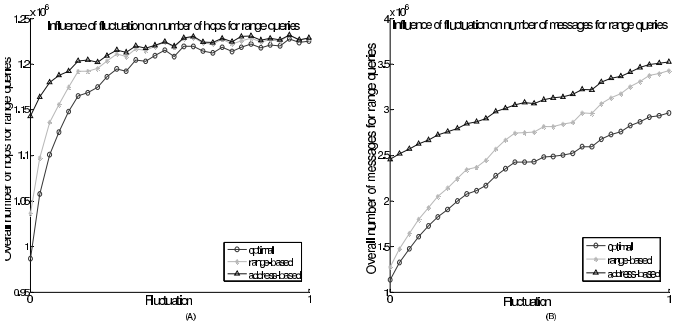

Fig. 21. Influence of SSV Value Distribution on Individual CLPs on Routing Cost

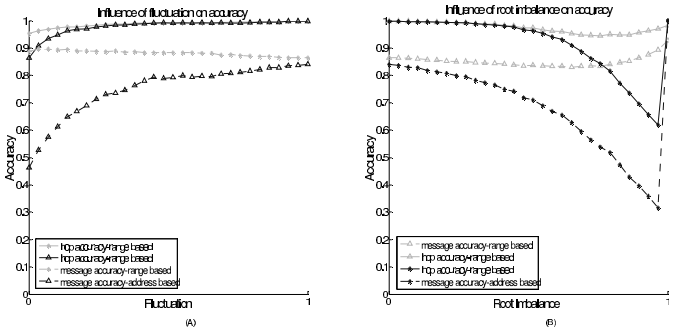

Fig. 22. Influence of SSV's Physical Distribution Pattern on the Accuracy of Routing Algorithms

E. In-depth Study on Range-based Solution

In this section, we first present the experiments on how SSV updates affect the performance of ranged-based routing algorithm. The key parameters have been varied are: (1) the *ratio* of number of updates to number of range queries, (2) the *fluctuation* between the updated value to the original value, and (3) the *granularity of segments* used in range-based algorithm. We also introduce a set of preliminary experiments on the effect of agent's behaviour.

1) Effect of SSV Update (Application Level)

An agent randomly updates any SSV whose value is in its region of interest. The SSV's value is updated with absolute offset (to the old value) randomly set conforming to uniform distribution with lower bound 0 and the upper boundary varied from 0.01 to 5. Furthermore, the probability that an agent generates an update query is set between 0.01 and 0.5.

Figure 23 gives an overall picture of the correlation between offset of updated SSV value and the ratio of update queries to the messages (update messages) needed for accomplishing update queries, in which three specific ratios of update queries are highlighted. The results indicate that the increase of both offset and the ratio of update queries lead to the generation of more update messages. Compared to the number of overall messages obtained in the previous experiments, the number of update messages is very small. Although this depends on the

specific configuration of the experiments, it still suggests that the overhead incurred by update query tends to be negligible in an environment with heavy load and traffic.

Figure 24 demonstrates the effect of the ratio of update queries to the routing cost and the breakdown of routing cost involved in update query and range query. The ratio makes significant impact on the number of messages/hops required by both queries. Additionally, Figure 24(A) shows that basically the address-based algorithm outperforms the range-based one where exist frequent update queries. Although the address-based algorithm generates more messages for range queries, it still incurs fewer messages in total due to the benefit in update queries. Figure 24(B) indicates that the number of hops is linear to the ratio of update queries and different routing algorithms do not make considerable difference in this aspect.

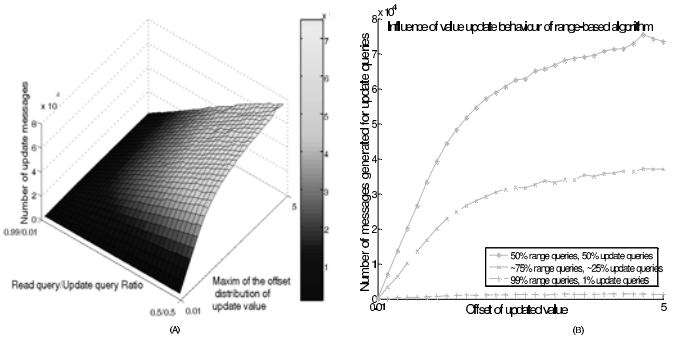

Fig. 23. Agent Behaviour's Influence on Number of Messages for Update Query

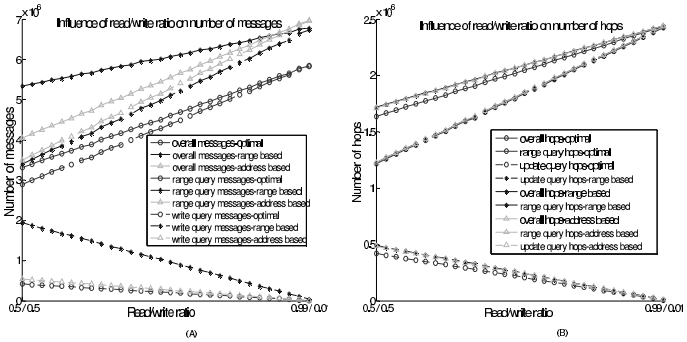

Fig. 24. Influence on the Ratio of Update Queries to Routing Cost

2) *Effect of Granularity of Segments (Simulation Level)*

The experiments and results presented in the section mainly concern the granularity of segmenting attribute value range for the range-based algorithm. The number of segments used to describe an attribute value range (segment number) varies between 1..200. A larger number means a more precise attribute value range description. The routing cost against segment number is reported in Figure 25.

The number of messages required by address-based algorithm is greater in most cases. Furthermore, the address-based algorithm excels in minimising latency only when segment number exceeds 100. Address-based solution should be independent of the granularity of segment by its nature. However, the address-based algorithm for this study forces a CLP to update all neighbours if an update query changes its attribute value range whereas the range-based algorithm is designed to only update those being affected (Section IV). It can be observed that when the number of segments becomes large enough, the range-based algorithm tends to update CLPs in a rather wide extent, and this will vastly deteriorate its performance. Figure 26(B) provides extra criteria to exhibit ranged-based algorithm's better adaptation of routing range queries to smaller segment. Figure 27 illustrates the overall difference (range-based minus address-based) of the two algorithms in number of total messages. Negative values denote the space for achieving better performance for adopting range-based algorithm, and vice versa for adopting address-based algorithm.

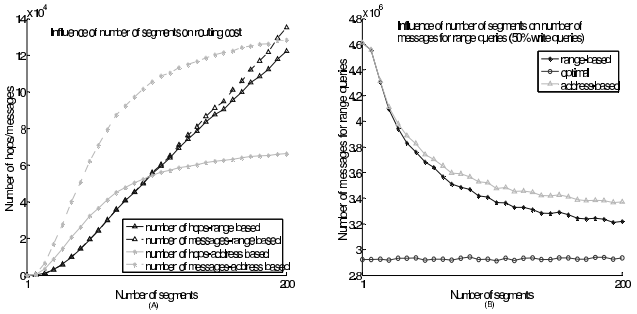

Fig. 25. Influence of Segment's Granularity on the Routing Cost

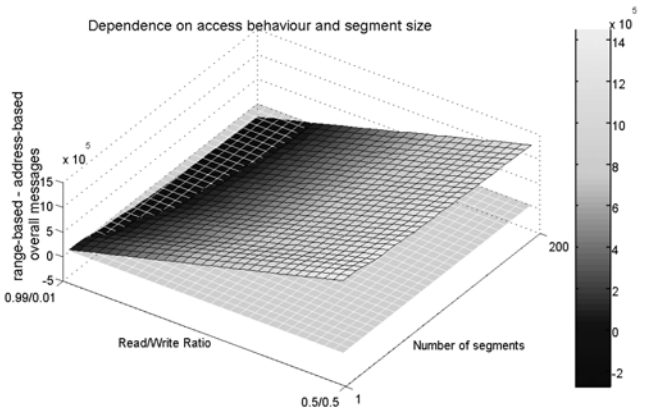

Fig. 26. Overall Difference in Message Number using Different Routing Algorithms

3) *Preliminary Test on the Effect of Agent's Behaviour (step size)*

In the experiments presented above, the environment contains 64 agents randomly situating on a 100×100 surface and 12,400 SSVs distributed either sparsely or concentrated in 9 central points (Figure 15). This suggests that in such a crowded environment no matter how far an agent moves (regionally or globally); there are always plenty of SSVs for an agent to access in its regions of interest. It is difficult to identify the effect of agents' behaviour under this circumstance. As far as this is concerned, we have carried out a batch of preliminary experiments with only 4 agents and one type of objects (1,550 SSVs of two types) to obtain a relatively spacious environment for agents and objects. Correspondingly, the size of CLP tree (depth = 2) is reduced to fit the agent number. In this set of experiments, an agent's behaviour pattern can be characterised by its activity scope and the speed at which it moves. For each

agent, its range of interest is 5.0 and its stepsize is varied between 0.1 (agent moves regionally) to 5.0 (agent moves globally). Two different settings of target distance, (1, 0) and (5, 2), are tested to mimic different speed of agent moving. The rest parameters have been set to default.

Figure 27 reports the effect of the size of an agent's activity territory in terms of number of hops and messages. Figure 27(A)&(B) report results for "quick" agents and (C)&(D) for "slow" agents. Basically the address-based algorithm incurs more latency while its overall cost is almost neural to the agent's moving pattern. When an agent moves more globally and quickly, the range-based algorithm becomes more costly. When an agent moves quickly (Figure 27(B)), it needs to generate slightly more messages than the address-based algorithm does.

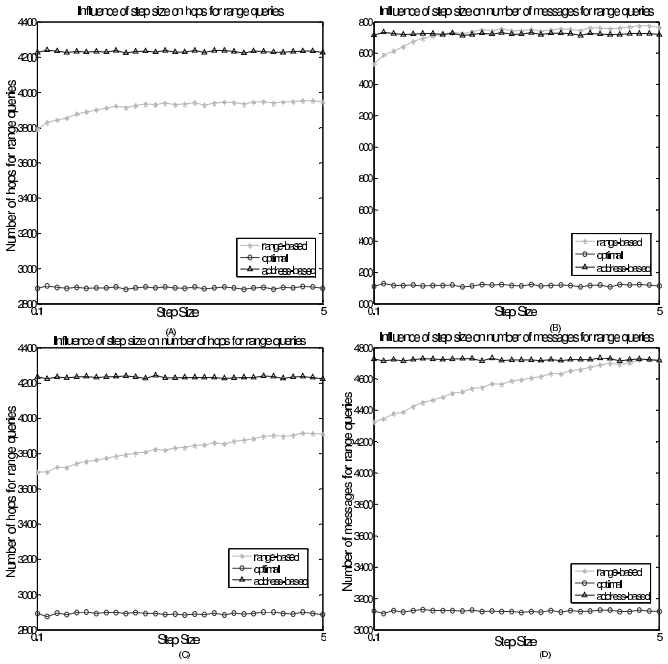

Fig. 27. Effect of Agent's Access Pattern

VI. Related Work

Jang and Agha developed a framework for distributed agent-based simulation [7]. They suggested dynamic distribution of agents to reduce the communication overhead in performing shared tasks and interacting with the environment. The approach also aimed at sharing load between agents. Significantly different to their work, our framework uses *SoI* as criteria to determine the optimal distribution of share state. Schelfthout et al proposed agent implementation patterns to describe generic solutions to design agents and multi-agent systems [25]. Shared state is modelled as shared state variables in their work. This approach is adopted in our framework.

As for data accessing in distributed environment, it is a vital issue to provide cheap and efficient routing (or locating) to desired data. Network routing issues has been heavily studied; numerous classic network routing algorithms have been developed for a router deciding on which output line an incoming packet should be transmitted [15]. In TCP/IP based networks, such as Internet, routers use routing algorithms to forward packet in correct path(s) over the network by referring to the address of the destination(s). Sometimes searching for an optimal path in a stationary network is already a difficult problem, therefore routing in mobile network will be much more difficult [15][15]. Hence, we prefer using a fixed infrastructure rather than exploring optimal routing algorithms to overcome the complication incurred by dynamic network structure.

As share state can be redistributed frequently, it demands a routing solution to locating moving (mobile) data in a possibly stationary network. For locating mobile objects, Steen et al proposed a scalable location service using a distributed search tree [26]. In the context of their approach, a mobile object is referred to as any component capable of changing locations in a network. The approach is address-based which binds an object's name and one more addresses where the object can be contacted. The location service is designed to handle objects with arbitrary migration patterns. Using hierarchical search tree, when each new object is registered, the leaf nodes store the addresses. By registering contact address in the smallest region [28] in which the object is moving, the approach only requires searching extremely short path to locate randomly migrating objects [26].

This suggests that it is viable to have a promising routing solution using address to identify distributed shared state and to locate mobile data items. Moreover, agent-based simulation often needs to perform range queries (see Section III.A) to meet the requirement for agents to sense the environment. There have been several previous investigations on range query for spatial databases [15][15][15], these works are focused on the optimisation of matching algorithms. The study on range query for distributed data has received little attention in distributed simulations [7]. The issue becomes much more complicated when the value and the physical distribution of data items both are dynamic. Range query targets on uncertain sets of data matching given conditions on the data's properties without having the knowledge of the locations of targets. The special feature raises the concern of relating the properties of data to the location information in the networked agent environment.

VII. Conclusion

In this paper, we have investigated some research issues in managing shared data in distributed simulations of multi-agent systems (MAS). A distributed simulation of MAS often involves a huge shared data with the interests of simulation models on shared data altering dynamically during the simulation. We have identified proper data distribution and efficient data accessing as the key issues to optimising the execution of MAS-based distribution simulations.

The paper has introduced the DS-MAS framework for supporting MAS-based distributed simulations. The framework uses the Communication Logical process (CLP) tree as an underlying simulation infrastructure. The framework

successfully facilitates dynamic data distribution in the CLP tree for approximating ideal Spheres of Influence. We have discussed different schemes for modelling shared data, i.e. composite type and simple type of Shared State Variables (SSV), and evaluated them from different aspects. Using simple SSV type can minimize the overhead for accessing different attributes of the same class of objects. In the case of performing range query for multiple attributes, using the composite SSV type is superior.

We have explored the design of efficient and effective data accessing solutions for locating and forwarding shared data and proposed two alternative solutions, namely an address-based solution and a range-based solution. The address-based solution locates SSVs according to their address information. In contrast, the range-based solution gropes after SSVs by looking up the attribute value range information along the paths to the destinations. The benchmark experiments have been performed to investigate the collective dynamics of the routing solutions in terms of cost and accuracy. The experimental results indicate that:

1) The SSV value distribution has a vast impact on the overall performance of both solutions. When the SSV values are distributed sparsely enough, both algorithms can minimize the latency of routing to the optimal case whereas they tend to generate a large number of messages (~20% more than the optimal case).
2) The distribution pattern of SSVs on the CLP tree is also a decisive factor to the performance of routing. The more evenly SSVs are allocated to the CLPs, the closer both solutions approximate the optimal case. The little the fluctuation of SSV values on an individual CLP is, the accurate the routing is. This denotes that routing can be optimised following the above rules.
3) Granularity of segment can considerably affect the routing of update query. The address-based solution is superior to the range-based one when segments are very precise. When range-based routing is adopted, precise segment means accurate routing. However, a precise segment leads to large overhead in dealing with update query.
4) In various scenarios, both algorithms have their own explicit advantages and drawbacks. It's desirable to have a hybrid scheme to adopt both algorithms in system design. In general, address-based solution provides optimal routing to ID query while range-based solution adapts better to different circumstances in range query.

The research work presented in the paper provides a novel method to analyse range query performance on distributed and dynamic data whose location and properties are changing constantly and unpredictably. This is an open challenging problem and particularly difficult to tackle using pure deterministic or analytic approaches. Besides agent-based systems, the method may also apply in studying other distributed systems, such as peer-2-peer communication system, distributed databases etc. For our future work, we need to further investigate the design and implementation of the appropriate routing algorithms into the existing DS-MAS framework. It is also interesting to probe the computational complexity of the routing algorithms at runtime. Another important issue is to have a throughout study on the impact of alternative load management mechanisms on the performance of routing algorithms.

REFERENCES

[1] An, N., J. Jin, A. Sivasubramaniam. 2003. Toward an Accurate Analysis of Range Queries on Spatial Data. IEEE Transactions on Knowledge and Data Engineering 15(2):305-323.

[2] Boukerche, A., C. Dzermajko. 2004. Performance evaluation of Data Distribution Management strategies. Concurrency and Computation: Practice and Experience 16(15): 1545–1573.

[3] J. S. Dahmann, F. Kuhl and R. Weatherly, "Standards for Simulation: As Simple As Possible But Not Simpler, The High Level Architecture for Simulation," *Simulation: Transactions of the Society for Modeling and Simulation International* vol. 71, no. 6, 1998, pp. 378–387.

[4] DIS Steering Committee. 1994. The DIS Vision, A Map to the Future of Distributed Simulation, Technical Report IST-SP-94-01, Institute for Simulation and Training, Orlando, Florida, USA.

[5] Epstein, J. M., R. L. Axtell. 1996. Growing Artificial Societies: Social Science From the Bottom Up, Brookings Institution Press.

[6] Faloutsos, C., I. Kamel. 1994. Beyond Uniformity and Independence: Analysis of R-Trees Using the Concept of Fractal Dimension, Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. pp 4-13, June.

[7] Fujimoto, R. M. Parallel Distributed Simulation Systems, New York: Wiley, ISBN 0-471-18383-0, 2000.

[8] Garey, M. R. and D. S. Johnson. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: Freeman.

[9] He, Q., M. H. Ammar, G. Riley, H. Raj, R. Fujimoto.2003. Mapping Peer Behavior to Packet Level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems. 11th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS. 2003

[10] IEEE 1516, Standard for High Level Architecture, 2001.

[11] Myeong-Wuk Jang and Gul Agha. 2005. Agent framework services to reduce agent communication overhead in large-scale agent-based simulations. Submitted to Simulation Practice and Theory.

[12] Li, J., R. Yates, D. Raychaudhuri. 2000. Performance analysis of path rerouting algorithms for handoff control in mobile ATM networks. *IEEE Journal on Selected Areas in Communications* 18(3):496-509.

[13] Liu, J., D. Nicol, B. Premore, A. Poplawski. 1999. Performance Prediction of a Parallel Simulator, Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99):156-164.

[14] Z. Juhasz, S. J. Turner, M. Gerzson and K. Kuntner, "A Performance Analyser and Prediction Tool for Parallel Discrete Event Simulation", International Journal of Simulation Systems, Science & Technology, vol. 4, no. 1-2, 2003, pp. 7–22.

[15] Kuhl, F., R. Weatherly, J. Dahmann. 1999. *Creating Computer Simulation Systems: An Introduction to HLA*. ISBN 13-022511-8, Prentice Hall, USA.

[16] Lees, M., B. Logan, R.Minson, T. Oguara, G. K. Theodoropoulos. 2004. Modelling Environments for Distributed Simulation, *1st International Workshop on Environments for Multi-Agent Systems (E4MAS), in conjunction with the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS04)*, New York, July.

[17] Logan, B., G. K. Theodoropoulos. 2000. Dynamic interest management in the distributed simulation of agent-based systems. *Proceedings of the Tenth Conference on AI, Simulation and Planning*. pp 45-50.

[18] Logan, B., G. K. Theodoropoulos. 2001. The distributed simulation of multi-agent systems. *Proceedings of the IEEE* 89(2): 174-186.

[19] Morse, K. L., M. D. Petty. 2001. Data Distribution Management Migration from DoD 1.3 to IEEE 1516, *Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications*. August.

[20] Oguara, T., D. Chen, G. K. Theodoropoulos, B. Logan, M. Lees. 2005. An Adaptive Load Management Mechanism for Distributed Simulation of Multi-agent Systems. *Proceedings of the Ninth IEEE International*

Workshop on Distributed Simulation and Real-Time Applications, October.

[21] Pagel, B., H. W. Six, H. Toben, P. Widmayer. 1993. Towards an Analysis of Range Query Performance in Spatial Data Structures, *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. pp 214-221, May.

[22] Perumalla, K. S. , R. M. Fujimoto, P. J. Thakare, S.Pande, H. Karimabadi, Y. Omelchenko, J. Driscoll. Performance Prediction of Large-Scale Parallel Discrete Event Models of Physical Systems. Winter Simulation Conference 2005.

[23] Pollack, M. E., M. Ringuette. 1990. Introducing the Tileworld: Experimentally Evaluating Agent Architectures.AAAI.

[24] Rajive, B., E. Deelman, T. Phan. 2001. Parallel Simulaiton of Lage-Scale Parallel Applications, The International Journal of High Performance Computing Applications, Volume15, No.1:3-12.

[25] K. Schelfthout, T. Coninx, A. Helleboogh, T. Holvoet, E. Steegmans, and D. Weyns, Agent Implementation Patterns, Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies (Debenham, J. and Henderson-Sellers, B. and Jennings, N. and Odell, J., eds.), 2002.

[26] M. V. Steen, F. Hauck, P. Homburg and A. S. Tanenbaum, "Locating Objects in Wide-area Systems," *IEEE Communications Magazine* vol. 36, no. 1, 1998, pp. 104–109.

[27] Sycara, K. P., "Multiagent Systems," *AI Magazine: The American Association for Artificial Intelligence* vol. 19, no. 2, 1998, pp. 79–92.

[28] A. S. Tanenbaum, *Computer Networks*, Fourth Edition. ISBN 0-13-066102-3, Prentice Hall, USA, 2003.

[29] G. K. Theodoropoulos and B. Logan, "A unified framework for interest management and dynamic load management in distributed simulation," in *Proc. 12th European Simulation Symposium*, 2000, pp. 111–115.

[30] G. K. Theodoropoulos and B. Logan, "An approach to interest management and dynamic load management in distributed simulation," in *Proc. 2001 European Simulation Interoperability Workshop*, 2001, pp. 565–571.

[31] A. M. Uhrmacher and K. Gugler. "Distributed, parallel simulation of multiple, deliberative agents," in *Proc. 14th workshop on Parallel and distributed simulation*, 2000, pp. 101–108.