# OUTLINE OF AN AGENT BASED APPROACH FOR A DYNAMICALLY DISTRIBUTED SERVICE

**[1]Prof. A.Damodaram, [2]I.Ravi Prakash Reddy**

[1]Prof. Dept. of CSE, JNTU College of Engg. Kukatpally Hyderabad
[2]Associate Professor, IT Dept. GNITS,Shaikpet Hyderabad-8

Email: irpr@gnits.ac.in

**ABSTRACT**

The concept of distributed computing implies a network / internet-work of independent nodes which are logically configured in such a manner as to be seen as one machine by an application. They have been implemented in many varying forms and configurations, for the optimal processing of data. Various benefits, e.g. speedup, scale up, enhanced reliability, resource sharing etc, are accrued on their optimal dynamic exploitation.

Agents and multi-agent systems are useful in modeling complex distributed processes. They focus on support for (the development of) large-scale, secure, and heterogeneous distributed systems. Research in this domain includes scalable and secure agent platforms, location services, directory services, and systems management. They are expected to abstract both hardware and software vis-à-vis distributed systems.

For optimizing the use of the tremendous increase in processing power, bandwidth, and memory that technology is placing in the hands of the designer, a Dynamically Distributed Service (to be positioned as a service to a network / internet-work) is proposed. This paper examines the rationale for, and features of, such a service. An agent approach with thread migration is recommended as the scheme of implementation. The basic mechanism underlying the scheme is discussed, along with the data structures that are migrated to implement the service.

**Keywords:** *System Software, Distributed Systems, Agents , Process Migration*

## 1. INTRODUCTION

Over the last two decades, the concept of distributed computing has been implemented in varying configurations and on diverse platforms. In current context, a distributed system implies a networked system of nodes in which a single application is able to execute transparently (and concurrently) on data that is, (or may be) spread across heterogeneous (hardware & operating system) platforms. The salient payoffs of distributed computing may be listed as

- Enhanced performance (in respect of both speed up and scale up).
- Resource sharing (in respect of data as well hardware and software processing elements).
- Fault tolerance and enhanced reliability.
- Serve as the basis for grid computing.

Several other relevant issues while assessing the relevance of distributed computing vis-à-vis the current computing environment and this paper are: -

- Interoperability in a heterogeneous environment will continue to be the dominating theme in future applications.
- Communication technology advances will continue to fuel the need for more bandwidth and enhanced Quality of Service specifications.
- The rate of increase in data processing and storage is greater than that of data transfer.
- Most users are reluctant to switch platforms, as are developers to switch technology paradigms.
- The individual behavior of the vast number of interconnected nodes based on individual workstations mandate that any

service acting upon them universally must be dynamic in nature.

- In many computer installations/complexes, a lot of state of the art hardware is not used round the clock. There are times when it is idle, or is under-utilized. When this happens, it may be used by other applications for processing purposes remotely. Networking enables this. Inter-networking further emphasizes the same.

In view of the above, it is forecast that distributed processing of applications and data will no longer be restricted to high end research and scientific applications, but will become as normal as any other service provided over an inter-network. The Internet and the Web themselves are a viable distributed computing domains. Distributed computing however, has yet to gain the type of proliferation mandated by enhanced rates of data processing as well as transfer.

To further the optimization of internet-works (including networks), by the use of distributed computing concepts, a Dynamically Distributed Service, analogous to e-mail, FTP, Voice over IP, etc, is proposed, which can be made available on demand, in an intranet/inter-network. The service will conceptually disintegrate an application into various constituents, process each of these simultaneously and on different nodes, and then integrate it again, after the benefits listed above have been accrued. This Dynamically Distributed Service (DDS) is at variance with distributed paradigms used till date, though *no changes to the underlying hardware or OS are proposed* in its implementation.

## Distributed Agents

The research areas of multi-agent systems and distributed systems coincide, and form the research area of *distributed agent computing*. Multi-agent systems are often distributed systems, and distributed systems are platforms to support multi-agent systems[1]. This is depicted in Fig 1.

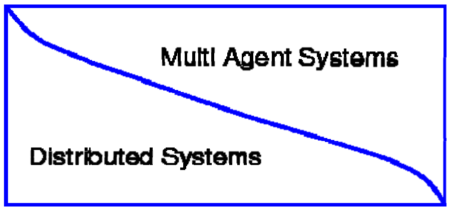

*Fig 1.* **Relationship between the research areas of multi-agent systems and distributed systems.**

*Agents* are considered to be autonomous (i.e., independent, not-controllable), reactive (i.e., responding to events), pro-active (i.e., initiating actions of their own volition), and social (i.e., communicative). Sometimes a stronger notion is added (beliefs, desired, intentions) realizing intention notions for agents. Agents vary in their abilities; e.g. they can be static or mobile, or may or may not be intelligent. Each agent may have its own task and/or role. Agents, and multi-agent systems are used as a metaphor to model complex distributed processes.

Both distributed systems and agents share the notion of 'distributed ness'. The area of multi-agent systems addresses distributed tasks; distributed systems addresses supporting distributed information and processes.

The area of *distributed agent computing* is the area in which both approaches intersect. Both can be synergized to further optimality.

## 2. RELATED WORK

A great deal of research has gone into the optimal design and scheduling of distributed systems. This paper focuses on two major issues -:

- A scheme for migrating smaller tasks to heterogeneous nodes within the network seamlessly by the use of agents.
- Setting the scenario for such a service, and resolving the issue of scheduling therein. The main burden of distribution must of course be shared by local nodes (to lessen the impact of communication distances), but no firm bounds are enforced.

Various open paradigms like CORBA (based on IDL), Remote Method Invoking (based on Java), and DCOM (distributed component object module by Microsoft) already exist, they are mainly devoted to brokering openness by distributed object computing (DOC)[2,3,4]. They make available a set of components, which may be used by applications across heterogeneous platforms to communicate with each other.

A commercial application that provides parallel execution of computing tasks in heterogeneous computing environments using autonomous agents has already been implemented[22]. It provides parallel computing by "breaking of a computer task into smaller tasks; assigning the smaller tasks to mobile agents and transferring the mobile agents to computing hosts; executing the mobile agents on the computing hosts.....determining network latency affecting transfer of data between computing hosts….. halting transferring mobile agents if the latency exceeds a particular threshold ….. monitoring utilization of computing hosts; halting transferring of mobile agents if utilization of computing hosts exceeds a threshold." The mechanics of implementation are by ***realistic thread migration.***

In a paper on Internet Distributed Computing (IDC) [21] Milenkovic et al forecast that the future evolution of Internet Distributed Computing will likely include proactive and pervasive computing. "We assume that the nodes willing to share a certain subset of their resources use mechanisms to announce their availability, possibly stating the terms of their usage to the rest of the distributed system or systems in which they are willing to collaborate." They feel that this may be achieved through resource virtualization and discovery by embedding intelligence in the network and creating self–configuring, self-organizing network structures. "Mindful of the successful Internet model built on existing standards and technology, we have sought areas that could provide reusable ideas, technology, or standards…. We postulate that such a system can be built as a combination and extension of Web service peer to peer computing, and grid computing standards and technologies."

Based on a preliminary analysis, the mechanics of our approach to modeling a distributed solution varies from the above, and are based on an effort to answer the following questions-:

- Should the focus be on load sharing or load balancing?
- Whether a dynamic scheme (with all its overheads) will prove more optimal?
- Should the scheme proposed be restricted to its applicability to the Web (and transitively to the internet) or be applicable to all networks and internet-works (truly pervasive)?
- Where in the overall system hierarchy should such a service be placed?
-

## 3. DYNAMIC vs STATIC DISTRIBUTION

Static distribution is considered more effective in a computing environment that deals with applications that can be portioned into uniform jobs requiring non-fluctuating resources. Furthermore these resources should be homogeneous, with similar operational characteristics (clock speed, architecture, bus speeds, buffer size, no of I/O channels etc). To be generic in its application, the service must be responsive to non-uniform problems. Also, the instantaneous state of availability of participating nodes is continuously varying. This must also be considered. Load balancing and load sharing also influence the decision. A distinction must be made here between load balancing and load sharing. Load balancing refers to the global load being balanced between various interconnected machines. The DDS however, is proposed to act as a service, and its aim is considered to be load sharing - or the distribution of an application running on a heavily loaded machine to other lightly loaded or idle machines Though many algorithms exist for both the techniques, load sharing in an inter-network, and load balancing to a limited extent with in the network is felt to be most appropriate. A dynamic distribution scheme is therefore considered to be optimal in the worst-case scenario of processing of non-uniform generic applications in a non-uniform, heterogeneous environment, not only with respect to the different *types*, but also similar types but different capabilities. This intuitive inference has to be investigated further.

Assume a network N of 'n' nodes T1, T2……,T-n. Suppose that an application 'A', running on T1 has to be distributed on the network n. Static allocation implies that a certain no of m (m<=n) nodes are available distributing the application 'A' ('A' is an instance of a uniform application). Dynamic allocation implies a random availability of node from zero to n. The number of node that will be available for the duration of execution of 'A' can be approximated to a set of random numbers by the Linear Congruential Method by applying the following recursive relationship :

$$X(I+1) = (aX(I)+c) \bmod m \quad i = 0,1,2,3\ldots.$$

The results obtained for the static allotment of n/4, n/2, 3n/4, and n vis-à-vis the dynamic allotment are given in Figure 2, 3, 4 and 5 respectively. It is seen from an analysis of the of the four cases that to viable in time cost terms, not more than approximately forty percent of the nodes of a network should be idle and participate in the scheme if dynamic distribution is to be resorted to. If the participant nodes in the network are greater than forty percent a static distribution scheme is more time effective. It is thus reasonable to assume that a the time for execution of an application will be optimized for a dynamic scheme only – as not more than fifty percent of the cumulative processing power of the number of nodes in a network will be available to an application for distribution. This leads us to the inference that any distribution scheme that is to act as a service to a network or an inter-network must be essentially dynamic in nature.

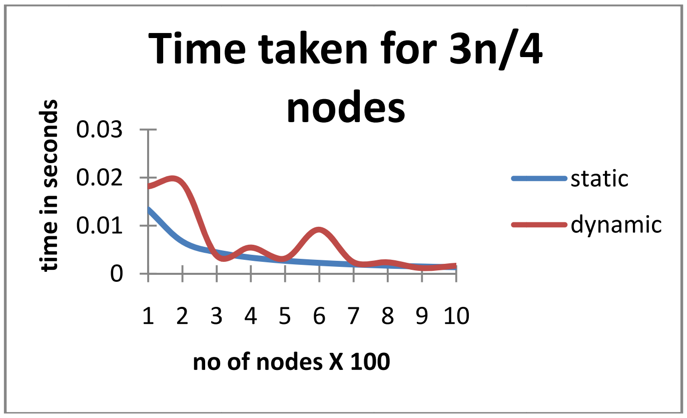

Fig 3

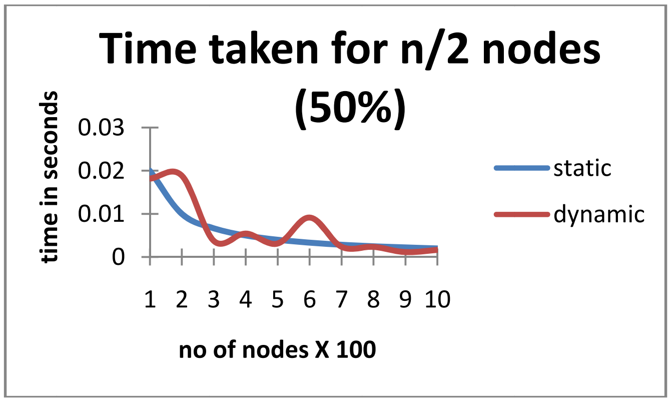

Fig 4

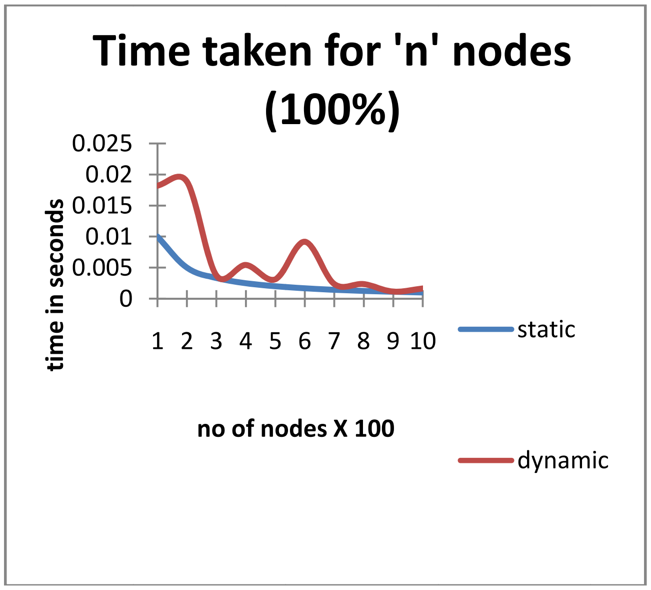

Fig 2

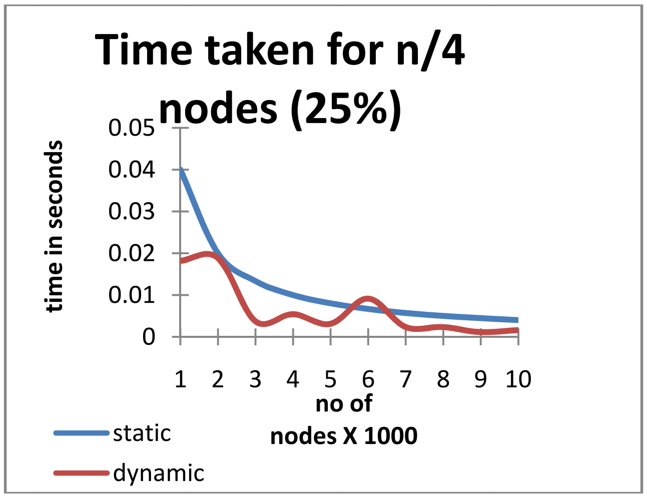

Fig 5

## 4. PROPOSED MODEL

We propose a model of an on-demand, voluntary Dynamically Distributed Service (DDS) to be used on an intranet / inter-network. The salient features of the service are -:

- Transparency. The most fundamental feature of the DDS is that once the appropriate options have been exercised, it will be fully transparent to the user in the discharge of its functions.
- No Reliance on Central Site. No reliance on a pre-determined central machine is thought necessary. The machine on which the job is initially being executed shall dynamically act as the logical processing center. The native OS on an option exercised by the user shall hand over the job to the DDS, which will proceed to migrate the various constituent threads (of the processes) to the participant nodes by use of agents. A 'thread' is used in the context of not merely a light-weight process but any unit of code that is capable of independent execution on a foreign node. The results of execution shall again converge on the source machine on the termination of execution.
- Hardware, OS, and Network Independence. The DDS will interface with heterogeneous platforms, much on the lines of a virtual machine [3].
- Local Autonomy. The participant nodes with in the DDS will be autonomous. This does not imply that processes of a job will not be shared or that they will depend on each other. They will. No site however will depend exclusively on another for the completion of its job. Thresholds based on time and space complexity are set determine which particular node is tasked with a particular process and for how long.
- Distributed Message Passing. Non-blocking, asynchronous primitives will be used. A 'send' or 'receive' will not result in the suspension of the job. Reliability shall be assured by the underlying protocol. Copies of the process shall be made before it is migrated. Buffer management shall be inbuilt. A standardized format for messages (common objects and data) shall convert the native code on any machine to that fit for migration, and translate into native code for another machine. This process shall be repeated for the return migration. Remote procedure calls (RPCs) are [4] relevant to the extent that they merely provide a special resource (hardware or software) to be exploited by a consumer thread.
- Use of Threads. Threads migration is used, as it is more suitable for asynchronous behavior. Most modern OS kernels support multithreading and interfacing threads would decrease the overheads[8,9]. A thread itself might be blocked, while the process goes on. The blocked thread itself would be communicated to participating nodes. Preemptive multitasking by means of priority allocation is featured. Global priorities of threads are maintained to avoid synchronization issues resulting from simultaneous context switching of threads of the same priority.

The block schematic of the DDS proposed is given in Fig 6. It is to be noted that the positioning of the DDS in the scheme shown above is to interface with, *not* replace the native OS. In this aspect it varies from other distributed operating systems like Sprite [4](which is meant to also function as a network operating system), Amoeba [10](in which the same kernel is implemented using remote procedure call using threads), and Andrew [11](which differentiates between client and server machines).

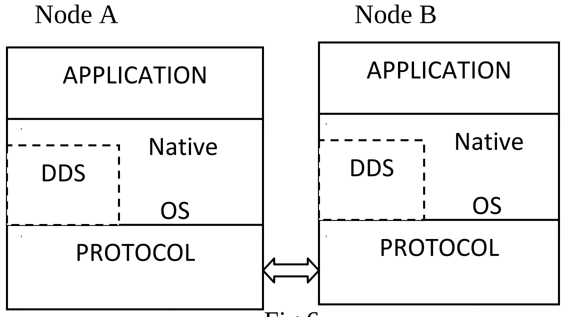

Fig 6

**The Positioning of the DDS**

**When, and by whom is Task Migration Initiated?**

The DDS proposed is to be positioned as a service on participant nodes in a network or an inter-network. The initial decision to invoke the DDS rests with the user initiating the job. Once it is invoked, thread migration commences transparently to participant with the help of agent-based algorithms. The algorithm will be based on load balancing implying that knowledge of load state of the participating nodes dynamically with in the network is necessary. Another feature of the algorithm will be the basis of the policy of when and where a thread (with its context) is to be migrated. It will depend on the state of the queue of 'ready' as well as the queue of 'suspended ready' threads. In the figure 6 below, Node A may be considered as the source node and node B, C, and D the participative target candidates. The process will be migrated to node B or Node C based on a global state. In this aspect it differs from conventional process migration wherein it is necessary to destroy the process on the source before it is migrated to some target system [4, 8,12].

**What portion of the job is migrated?**

The model proposes the replication of the state of the process, and the movement of a thread from the source to the target machine. The process image must be moved. The native OS will handle the hardware of the target node. Only those constituents will be migrated which are essential to the running of a thread of the process. A scheme of transfer of data residing in virtual address space on demand is followed. Though this lessens the amount of initial data transfer but does increase latency due to possible increased number of transfers. The issue is handled differently by various distributed platforms. Java interprets into byte code for the Java Virtual Machine [3]. The OMG object model establishes a client server relationship between objects – the client can transparently invoke a method on a server, which can be on the same node or across a network[2]. Andrew, a distributed computing environment developed at Carnegie Melon University, constitutes a partitioned filename system (between local and shared partitions), which supports the sharing mechanism among participant nodes [12]. The rationale for devising a varying ad hoc scheme for the proposed DSS is:

- The main thrust is on speedup and scale up - not on the use of remote functions.
- The DSS is intended to interface with the native OS – not the hardware. It relies on the interoperability offered by a reliable for execution on heterogeneous platforms.
- It is *not* an OS. It merely interfaces with it. File, memory, I/O management etc are still the responsibility of the native OS.

It is thus envisaged that the image of process will be migrated to the destination node – processed - and the results subsequently be reflected back to the source node. The process image of the DSS (state) is a data structure, which would comprise of the following: -

- Thread code to be executed – an interface will map the code of the source machine to the DDS data structure, which will reverse the process on the target machine.
- Thread Identification.
  - Thread id.
  - Process id.
  - Parent process id.
  - User id.
- User visible registers' contents.
- Process visible registers and memory contents (both constant and variable).
- List of open files.
- Priority of threads - this is to resolve conflicts between processes from two source machines onto the same target machine. As mentioned earlier, an incoming process will pre-empt no user process already executing on a target machine, unless the user of the target machine has explicitly authorized such pre-emption.
- System stack – this will store parameters and calling addresses for procedures to be called by the system. Since system calls are specific to the native operating system the reference to the system call will have to be replaced by its' object code. Thresholds abandoning the migration in case the overhead exceeds the speedup accrued will be set.
- Address references – This will contain pointer references to file names, routines, segments/routines that describe virtual memory assigned to the process. The pointer references must be resolved substituted by the actual contents that are pointed to, and subsequently packaged before migration.

- Internal Job Structure – A process may relate to another process already in some state/execution. The identifiers of that process will also be included.
- Thread context.
- State of I/O counters if the migrated thread is accessing them.

Fig 7 depicts the state of the system (identified by nodes A, B, C, and D) before the migration is to commence, as also the conditions under which it will take place. On an over simplification, assume that Node A is overloaded is running with Process A and a State $S_A$. It has various threads running concurrently. Though each thread would have its own state, assume that all the threads of Process A have a common state $S_{A.}$ Threads $T_{A2}$, $T_{A3}$, and $T_{A4}$ are migrated to nodes B, C, and respectively. The cumulative execution time of the four threads on four different nodes A, B, C, and D (inclusive of the overheads of the DDS) would be less than the execution time on the same node A. The DDS would interface with the native OS by abstracting the process context (and its constituent threads) for the network protocol stack. In this aspect it differs from CORBA [2], (an architecture to further openness in heterogeneous distributed systems), which seeks to make use of remote objects through its broker architecture. The DDS seeks to break up a job, disintegrate it, process it, and subsequently integrate it again.. Such a scheme is more relevant for a process intensive rather than I/O intensive job. It is obvious that it is as vulnerable to the limitations of available bandwidth as any other distributed system, the main difference being that it will add on to the latency of the native OS. The decrease in overall processing time of a process intensive job and the enhanced availability of networked computing power result in an overall improvement in scale up as well as speed up of the job. In this regard it will offer the advantages of parallel processing with the following major differences [4 & 13]:

- The nodes may be geographically far apart.
- Even less powerful nodes (clock speed, cache, main and secondary memory etc) may act as constituents.
- A loosely coupled scheme is proposed.

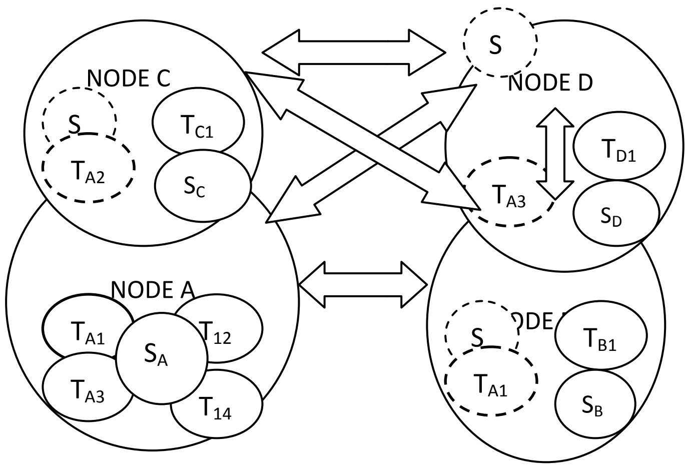

Fig 7– Thread Migration

## 5. AGENT BASED SCHEME

Fundamental to a distributed data processing is the activity of migration of code, or the transfer of sufficient amount of the state of a process (comprising of many threads) executing on one machine to another set of machines in such a manner that it can be executed on either. Since a thread is a lightweight process, and capable of asynchronous execution, it has been chosen as the unit of migration. Currently operating systems support multithreading – our perception of thread migration however, is the asynchronous, non-blocking migration of any unit of that can execute independently. The DDS to function effectively must have real time access to the following:

- Global 'awareness' of the state of computation in the network. Only then can the load be distributed optimally. 'Awareness' includes *what* is to be executed, and *who* is available to execute. It must also have the ability to decide on an optimal allocation scheme. Apart from all this, information about special resources must also be held. This function is intended to be carried out by a 'scheduling agent'.
- A mechanism for converting the application into a form that can be executed by other constituent nodes which might be heterogeneous in their computing orientation. The function must be performed at both the receiving and sending node. This function is intended to be carried out by the 'interfacing agent'.

- A mechanism for brokering the migration of a job, and carrying out this migration through a suitable message passing scheme. This would not hinder the use of RPCs. The interfacing and migration is proposed is to be carried out a level lower than that of RPC implementation and there is no bar on the use of the scheme by nodes hosting software resources, as long as the thresholds are adhered to (processing benefits are accrued). Further ad hoc protocols and constructs would be used for the DDS of a source node to interact with its peer DDS in the destination node. This function is proposed to be carried out by the 'dispatch agent', in coordination with the interface agent.

The scheme, though oversimplified, is similar to the Stealth Distributed Scheduler[14] – it aims to use the largely under utilized capacity of computing nodes in a network. Foreign tasks can be executed in a node even when under by the owner of that node. Owners are insulated from excessive resource consumption by foreign tasks by a prioritized and pre-empted resource allocation scheme. The difference lies in the DDS being a service – each node is a willing participant and there is price involved – to be either paid or received. Also, the DDS is based on thresholds. If adequate service is not rendered, the target node on its own completes the task. In the worst case the only overhead is wasted bandwidth. The thresholds are to be built in as Quality of Service parameters into the service.

The sequence of actions , is as follows: -

- 1 – A node opts to participate in the scheme. The Scheduling agent is informed.
- 2 – The global load state of the network is updated. The scheduling agent not only keeps track of the availability of nodes for migration, but also keeps a record of the node to which migration would be optimized.
- 3 – A particular node has exceeded its threshold of load, and the process of thread migration is initiated.
- 4 – The interface agent starts preparing the executable unit of migration – to include the thread code and its state. As soon as it does so, it starts an internal counter for checking the overhead time consumed. The scheduling agent informs the dispatch agent of the address of the most preferable node.
- 5 – The dispatch agent of the source sends a message to the destination whether migration can begin on that node.
- 6 – The interfacing agent completes the packaging of the executable unit of code.
- 7 – The dispatch agent makes use of a reliable protocol for sending the executable unit. It dispatches it only after receipt of the confirmation from the destination node.
- 8 – On receipt of the execution unit by the destination node the interfacing agent of the destination node decodes it and hands it over to the destination node's kernel for processing.
- 9 – As soon as the execution is over, it is once again handed over to the interface agent and the dispatch agent for its move back to the source node.

In the specific case when $T_1$ is migrated from node A to node B and is executing, it might occur that a process native to node B suddenly becomes active. In this case it becomes mandatory to relocate $T_1$. Process $P_1$ is immediately suspended on node B and repackaging and dispatching of its' process image by the scheduling agent of the source begins. This is only in case specified thresholds have not already been exceeded. The point to note here is that instead of returning to node A, it is sent to some other pliable node with in the domain. The time taken for it to execute and return to node A is limited by an upper bound, but the migration is still supervised by the original source node.

## 6. SYNCHRONIZATION ISSUES

Consistency requirements mandate event ordering which in turn implies awareness of a global state. The instantaneous global state of any distributed system requires real time awareness of all processes, memories, registers, I/O channels, files, communication channels etc, within the bounds of the distributed system. In the absence of a system-wide common clock, that regulates the timing of events, this is unrealistic, if not impossible, as the global state cannot be obtained by merely putting together the local states of each node. Event order based on time precedence is not viable, and thus an event has to be associated with the serial chronology of the message under which the thread of the associated process was migrated. The well established technique of timestamping is used to handle the issue. Stallings[4] treats the topic adequately. What has been the general consensus till date [15] is that a process when it is migrated destroys itself on the source node (node A in Fig 7) to recreate itself in the destination node (node B). The DDS however, does not destroy the process – it migrates one of the threads of the process, and pushes the thread into the suspended state – either in main or secondary memory. For inter-thread synchronization, a variant of the normally accepted scheme in [4] is used. The message is sent in the form

{m,T,q,k,j,p,t,n} where

m → Contents of message

T → Timestamp of message initialized to '0'

Q → Numerical identifier of original source node (this will remain constant for the period of distribution of thread)

k → Numerical identifier of subsequent source nodes

j → Numerical identifier of the job

p → Numerical identifier of the process

t → Numerical identifier of the thread

n → Numerical identifier of the node serial relative to when it was migrated.

The DDS in each node sets up a local counter the moment it is invoked by the user. This counter serves as a clock for that particular session for that particular. The interface agent of each node applies the following procedure: -

```
Message_send // For a unique thread for the source node A
        {local_clock ++;
          T = local_clock;}
Message_receive // For a unique thread for destination node B the
        { if T>local_clock then local_clock=T+1
          else local_clock++;
              n=n++;}
```

This resolves the problem of ordering a unique thread as it is distributed from node to node by setting the condition that a message thread $t_1$ has preceded $t_2$ if $T_{t1} < T_{t2}$ for *a unique process.* As the process itself is not destroyed on the source node, the numeric identifier of the source node *n* will always be zero. The monitor will keep track of where the thread is at particular instant by updating the value of *n* dynamically and mapping it to *q*. To resolve precedence between threads having the same time stamps but belonging to different processes, the numerical identifier *q* (the owner node of the thread) will be used. The node identifier is a hashed function of the IP address of the node, and is computed and maintained by the interface agent.

The major issues relating to concurrency are the enforcement of mutual exclusion as well as the avoidance of deadlock and starvation amongst the distributed threads of a job. These issues vis-à-vis the DDS have to be resolved amongst the *distributed threads* only as only these threads are being migrated.

**Mutual Exclusion, Deadlock, and Starvation**.

The issue of mutual exclusion is resolved between threads by the interface and dispatch agents. The thread makes a request by a message {m,T,j,p,t} to the interface agent (through the kernel) whenever it requires access to a critical section(the notations are as per those listed above). The interface agent checks with the source kernal (via the dispatch agent) to check the state of the critical section. If it is, the critical section is allotted to thread *t*. If the critical section is in use, it is puts in a FIFO queue.(Priority based FIFO queue can also be used) The placing of the thread in the queue itself is based on the value of its timestamp. The scheduling does however put an upper limit on the time that the thread can consume the resource, subject to the condition that there other threads in the queue for that particular resource. The DDS does *not* interfere or override with the native OS having charge of that resource – but merely interfaces with it (if invoked). There is still the issue of coordination between the interface agents of different nodes on resource bids. How does a host kernel which is allocating a resource(through its native interface agent) know for certain that there is no thread before time $t_i$ (at say $t_{i-1}$), of the same process that was to be allotted the resource before $t_i$? The monitor of the destination node going to the source and confirming that there is no prior thread outstanding other than the current one that is requesting the resource resolves the issue. File consistency, as well global variable values integrity is maintained by a uniform timestamping scheme. Access to file and global variables can be had only by entry into critical regions through the appropriate DDS system calls and native OS interface.

Deadlock can be avoided by many ways. Let us examine a typical one here,the Chandy-Misra-Haas algorithm(Chandy et al.,1983). In this algorithm, processes are allowed to request multiple resources(e.g. locks) simultaneously instead of one at a time. By allowing multiple requests simultaneously the growing phase of transaction can be speeded up considerably. The algorithm is invoked when a process has to wait for some resource. At that point a special probe message is generated and sent to the process (or processes) holding the needed resource. The message consists of three numbers:the process that just blocked, the process sending the message, and the process to whom it is being sent .When the message arrives, the recipient checks to see if it itself is waiting for any processes. If so, the message is updated, keeping the first field but replacing the second field by its own process number and the third one by the number of the process it is waiting for. The message is then sent to the process on which it is blocked. If it is blocked on multiple processes, all of them are sent(different) messages. This algorithm is followed whether the resource is local or remote. If the message goes all the way around and comes back to the original sender, that is, the process listed in the first field, a cycle exists and the system is deadlocked. There are various ways in which the deadlock can be broken. One way to have each process add its identity to the end of the probe message so that when it returned to the initial sender, the complete cycle would be listed. The sender can then see which process has the highest number(or lowest priority), and kill that one or send it a message asking it to kill itself. Starvation of a resource is not possible because the monitor is able to suspend the execution of a thread if it exploits a resource beyond a pre-determined bound.

## 7. PERFOMANCE MEASUREMENT

The performance of the DDS depends directly on the extent to which delivers on the Quality of Service parameters laid down for the environment in which it functions. The rationale for enunciation of the same is explained in Fig 9.

For a particular thread T's migration, let

$t_i$ = latency time for interfacing T (in both the source and destination)

$t_s$ = time required for scheduling T's migration

$t_d$ = latency due to the network (dispatch agent)

tc = Time taken for T to execute on a single node with 100% CPU allotment

$t_m$ = Time delay due to multiprocessing

Then for the DDS to be viable, the total time of execution by migration should be less than that taken by a single node by multiprocessing, ie $t_m > t_i + t_s + t_d$

## 8. CONCLUSIONS

The DSS proposed has attempted to introduce dynamically distributed service as inherent to an inter-network as FTP, TELNET, e-mail, chat etc. The rationale for - and the main features of - the basic scheme have been described. Process intensive applications will be the main beneficiaries of the scheme. The outlines of the resolution of the basic issues of thread migration, mutual exclusion, deadlock, and process synchronization to serve as a basis for the service have been introduced. It is felt that as technology continues to hold good on its promise of enhanced data processing and transfer rates, the service benefits will overcome the overheads of the DSS.

Any viable service in an open inter-networking environment survives by initially identifying its underlying components, interfaces, and protocols, as well as how best to adapt to the topology in which it is used. The interfaces are especially important. This paper has introduced the policy for the DSS. The mechanism of interfacing the modeled DSS with a native OS is the next logical study domain, followed by an analysis of how the TCP/IP suite can be optimized for reliable thread transfer. The service may be extended to distributed databases also, but fragmentation independence issues will have to be addressed. Multimedia, Data Mining, & GIS constitute another exciting application domain. Simulation models have to be applied to forecast the efficacy of the DDS. A reliability analysis [16], evaluating the extent and conditions under which is most beneficial will also have to be carried out. Qualities of Service parameters have to be defined and quantified by stochastically.

## REFERENCES

[1]. http://www.dsonline.computer.org

[2]. "The Common Object Request Broker Architecture (CORBA) and its Notification Service" by Gupta and Kar, IETE Technical

Review, Jan-Apr 2002, Vol 19, Nos 1&2, pp 31-45.

.

[3]. Java 2 – The Complete Reference – by Peter Naughton and Schildt (2000) Tata McGraw-Hill

[4]. Operating Systems by William Stallings, Pearson Education

[5]. US Patent Application 20020156932 Kind Code AI by Marc Schneiderman dated 24 October 2002 for "Method and Apparatus for providing parallel execution of Computing Tasks**"**

[6]. "Towards Internet Distributed Computing", by Milan Milenkovic, Scott H Robinson, Robknauerhase, David Barkai, Sharad Garg, Vijay Tewari, Todd Anderson, & Mic Bowman , IEEE Computer May 2003 pg 38-45

[7]. "Allocation Aspects in Distributed Computing Systems" by Vidyarthi, Tripathi, and Sarker, IETE Technical Review, Nov-Dec 2001 pp 449-454

[8]. "Grid Computing : A Vendor's Version" (2002) – URL : http://dsonline.computer.org/0206/departments/new.htm

[9]. "Distributed Operating Systrems" by Andrew S Tannenbaum

[10]. "Operating System Concepts" Silberschatz & Galvin (2000) published John Wiley & Sons Inc.

[11]. "Application of Parallel and Distributed Data Mining in E-Commerce", IETE Technical Review Aug 2000 pp 57-60 by Beg and Ravi Kumar

[12]. "Advanced Concepts in Operating Systems" by Mukesh Singhal & Niranjan K Shivratri, Tata McGraw Hill Publishing 2001

[13]. ".NET vs JAVA" IT Today Sep 2002, pgs 62 – 75

[14]. "Reliability Evaluation of Distributed Computing Networks 2-Mode Failure Analysis" by S Selvan, Moinuddin, and KV Ahmad, IETE Technical Review Jan Feb 2001, Vol 18 pgs 45-51

[15]."Transparent Process Migration in Sprite Operating System" - URL :http://www.cs.berkeley.edu/projects/sprite/sprite.papers.html

[16] ."Load Balancing in Distributed Systems (1999)"- URL : http://www.ibr.cs.tu-bs.dc/projects/load/#PAPERS

[17] ."End System Optimizations for High Speed TCP" IEEE Communications April 2001 Vol 39 No 4, pp 68-76 by JH Chase, AJ Gallatin, and KG Yocum.

[18] ."Fast Dynamic Process Migration" ICDCS – 1996 –URL: http://www.cs.umd.edu/users/kritchal/opt.html

[19]. "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real Time Systems" by T Abelalzaher, Dar – Tzu Rand, & Kang C Shin, IEEE Transactions on Software Engineering, Vol 23, No 12, December 1997.

[20]. "Algebra of Communicating Shared Resources" (1999)- URL : http://www.cis.upenn.edu/~rtg

[21] ."Interfacing Wide Area Network Computing and Cluster" –(2000) Sumalatha Adabala http://citeseer.nj.nec.com/adabala00.interfacing.html

[22] ."Real Time Information Delivery for the Extended Enterprise" - URL: http://www.itworld.com/Net/1756/CISCO/010302/

[23] ."Optimising Heterogeneous Task Migration in the Gardens' Virtual Cluster Computer (2000)" –URL : http://citeseer.nj.nec.com/323338.html

[24] . "Managing IT", Computers Today, March 2002 pp38-42.