Automatic Resource Management in Multi-site Mobile Computing (Invited Paper)

Qinghong Shang†*, Munehiro Fukuda‡, Michael B. Dillencourt* and Lubomir Bic*
† School of Computer Science & Engineering
University of Electronic Science and Technology of China, Chengdu, Sichuan, China, 610054
‡Computing and Software Systems, University of Washington, Bothell, WA 98011, USA
mfukuda@u.washington.edu
*Donald Bren School of Information & Computer Sciences
University of California, Irvine, CA 92697, USA
{qshang, dillenco, bic}@ics.uci.edu

ABSTRACT

This paper focuses on automatic resource management in mobile computations that span multiple sites. We extend an existing static resource management system to a dynamic management system by providing a five-layer resource management mechanism. This mechanism is designed to support a Navigational Programming system called JaMes, which is a Java-based programming system based on self-migrating threads. The top two layers provide a top-down mechanism which allows JaMes to request resources based on performance criteria specified by the application programmer. The bottom three layers provide a self-organizing bottom-up mechanism that provides current information about the available systems resources. We illustrate the utility of this multilayer architecture with a classical matrix multiplication example executing in a multisite environment.

Keywords: Distributed parallel computing, mobile agents, multi-site resource discovery, resource based migration, Java-based programming systems.

1 INTRODUCTION

There are two approaches to dynamic resource allocation currently used in grid computing and cloud computing: (1) the top-down self-adaptive approach and (2) the bottom-up self-organizing approach [1]. SASO 2007 defined them as follows: self-adaptive systems evaluate their own global behavior and change it when the evaluation indicates that they are not accomplishing what they were intended to do. In contrast, self-organizing systems are composed of a large number of components that interact according to simple and local rules [2] to present a higher-level view of the resources.

This paper presents a five-layer approach to dynamic resource management for JaMes[3]. JaMes, short for Java MESSENGERS, is a Java-based programming system for Navigational Programming (NavP) [5,9]. Using JaMes programmers can invoke functions on remote nodes and also communicate (by ship and receive) with other nodes. JaMes applications operate within a logical network. JaMes only supports a static mapping of this logical network to the underlying physical network. To provide for dynamic resource allocation/management, we provide a five-layer support system that can take advantage of a dynamically changing resource environment. With this extension JaMes can request and utilize the set of resources best suited to its current needs according to criteria specified by the application programmer.

The five-layer model described in this paper represents a hybrid between the top-down and bottom-up approaches. The model is loosely based on the field-based routing concept described in [4], where each client message hops to a desired server along the steepest gradient of a service potential field in the same manner of an electrical potential field in physics that moves an electron, (i.e. a message in field-based routing) toward the electrically steepest gradient. Analogous to field-based routing, a sentinel agent can determine the desired computing nodes by following the steepest gradient of a potential field corresponding to the current status of the available computing resources. To create this field, an agent on each node monitors its local computing resources and broadcasts its resource information to its neighbors, so that the information is disseminated throughout the network in a heartbeat fashion.

After presenting an overview of JaMes in Section 2, we describe the five-layer model in detail in Section 3. Section 4 describes how the top-down and bottom-up portions of the model interact to provide a hybrid system. Section 5 illustrates the model using matrix multiplication in a multi-site environment. We then discuss some related work and provide some conclusions.

2 RESOURCE MANAGEMENT IN JAMES

JaMes[3] is a Java-based parallel programming that incorporates the principles of Navigational Programming (NavP). The advantages of NavP over Message Passing (MP) and Distributed Shared Memory (DSM) have been discussed in previous work [5,6,9]. In essence, MP can be difficult to use and DSM does not scale well.

2.1 Basic features of JaMes

JaMes applications are Java programs that can take advantage of the following additional features provided by the JaMes runtime environment.

Self-migration: Methods may be invoked remotely and asynchronously, meaning that the called method does not return a value and the caller does not wait for the called

method to complete. This provides the basis for migration. If the caller dies, this corresponds to a *hop* operation. If the caller continues, this corresponds to a *clone* operation.

Send/receive: JaMes incorporates sending and receiving of data while remaining faithful to the principles of Navigational Programming. The basic mechanism uses a *ticket*. When data is sent, the sender specifies the destination node and a ticket name. To receive data, a process specifies the ticket name, rather than the identity of the sender. This may be done either synchronously or asynchronously: the receiver may either block until a shipment with the given ticket name is received, or periodically check for the shipment.

Non-preemptive scheduling: JaMes uses a non-preemptive discipline, so that programmers do not need to worry about synchronization. On each node, there is only one executing thread running at any time. An executing thread cannot be preempted: the only way it can be blocked is when it blocks itself by issuing specific commands such as wait(), yield(), or a blocking receive request. Non-preemptive scheduling eliminates the need for explicit critical sections when accessing shared variables and reduces context-switching overhead.

Priority-based scheduling: JaMes supports priority-based scheduling. Since JaMes uses its own scheduler rather than the Java thread scheduler, the number of priority levels does not depend on the underlying operating system. The priority can be used to control both the order of computations that hop from another node and the choice of threads to be run when another thread gives up the CPU.

Logical-to-physical node mapping: JaMes applications are written to a fully-connected logical network, the size of which is defined by the application. Each node is referenced by its rank. This permits the application to be written in a manner independent of any specific hardware configuration.

2.2 Support for Dynamic Resource Management

JaMes is currently supported by only a static configuration scheme as shown in Figure 1(a). The mapping of logical nodes to physical nodes is set when JaMes starts up and cannot be changed. Specifically, a startup program is run on a set of nodes consisting of a master node and a statically designated set of individual nodes, listed in a configuration file. When a node registers with the master node, it is assigned a logical rank number by the master node. The rank number is nondeterministic as it depends on the order in which the UDP messages arrive at the master node.

This static approach can be improved in three fundamental ways:

1. **Application-specific mapping:** The logical-to-physical mapping for a particular application should be set when the application starts up rather than when the JaMes system starts up. This permits different applications running under the same instance of JaMes to use different mappings.
2. **Dynamic resource availability:** When the logical-to-physical binding is delayed until the application starts up, the mapping can take advance of the most recent status of the system. For example, nodes with the lowest CPU utilization or the largest amount of available memory may be chosen.
3. **Multi-segment network:** Since JaMes assumes a fully connected network, additional support is necessary to allow applications to run in multi-segment networks where the only connections between segments are the gateways.

Figure 1(b) illustrates the dynamic configuration management scheme, using the 5-layer architecture described in the following section.

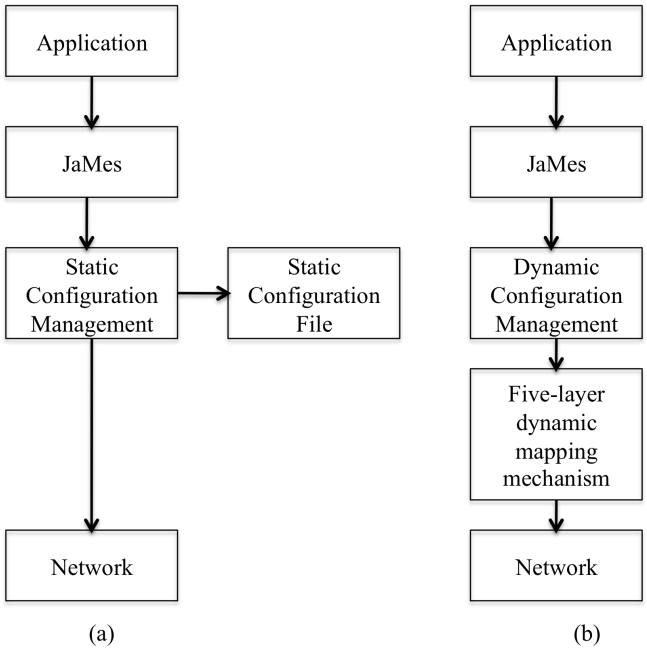

Figure1 (a) Static mapping mechanism (b) Dynamic mapping mechanism for JaMes

3 DYNAMIC RESOURCE MANAGER ARCHITECTURE

The 5-layer model shown in Figure 2 facilitates the dynamic mapping and resource based migration on multi-site environment.

3.1 TCP-Link-Assisted Inter-Segment UDP-Broadcast Space

The lowest layer is a TCP-link-assisted inter-segment UDP-broadcast space. Since UDP broadcast is normally limited to within a single segment, additional administrative support, such as IGMP, is necessary to allow broadcasting across multiple segments. Our implementation facilitates application-level inter-segment UDP by establishing a TCP-link between representative nodes of each segment which

allow relaying intra-segment UDP-broadcast messages among the segments. This layer is established as follows:
1. Each network segment chooses a representative node that runs a UDP-relay daemon locally.
2. Each UDP-relay daemon contacts a shared rendezvous point, (e.g., a common FTP server) to upload its IP address, to download the IP addresses of all other remote segment representatives, and to establish a TCP link (through a ssh tunnel) to each remote representative.
3. The UDP-relay snoops all intra-segment UDP-broadcast messages and relays them on each of the emanating TCP links to the remote representatives, while receiving messages from these representatives and broadcasting them to the local network segment.

3.2 UWAgents

The second layer is the UWAgents mobile-agent execution platform. A separate daemon process runs at each node. Its role is to exchange agents with other nodes and to run their code. Details about the UWAgents system can be found in [7].

3.3 Potential Field Agents

The third layer consists of Potential Field Agents (PFAgent). One PFAgent is launched at each node. Each PFAgent periodically measures the latest performance of its local computing resources including CPU power, memory space, disk size, network bandwidth, and their current availability. All of these are recorded in each PFAgent's internal resource table and are broadcast in a UDP message within the local network segment and relayed to remote segments. Each performance measure forms its own potential field. In response to a resource request from a user process, each PFAgent uses the specified field to find the nodes that best suit the needs of the process.

3.4 Sentinel Agent

The fourth layer is the sentinel agent. The sentinel agent is launched by the configuration management layer (layer 5) on the node where the JaMes application is running. The sentinel agent finds the best currently available nodes, based on criteria passed down to it from the configuration management layer. It does this by querying the PFAgent on the same node. The PFAgent is responsible for monitoring performance measures such as CPU utilization, memory usage, and CPU power and disk size. In addition, the PFAgent may specify more specialized criteria, such as the availability of certain libraries or software tools. Given the multitude and unpredictability of such possible queries, it is not practical to maintain a complete up-to-date list. Therefore, the PFAgent gathers such information only when prompted by requests from the sentinel agent.

3.5 Configuration Management

The fifth layer is the configuration management layer, which is responsible for mapping logical nodes to physical nodes dynamically and automatically. Because all node information is transparent to programmers and applications, only logical rank numbers are used in the code as the destinations for computation migration and data exchange. The logical numbers are bound to physical nodes at run time

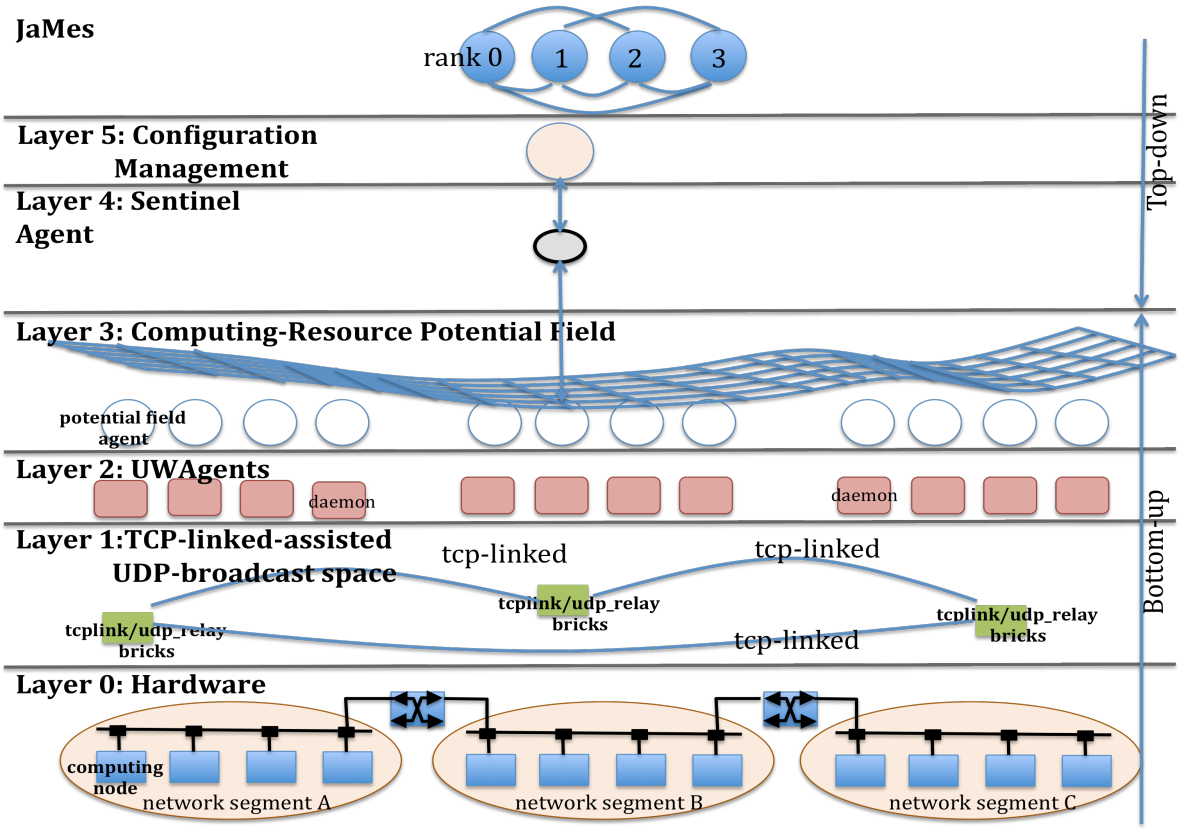

Figure2 Execution layers used for field-based process dispatch and migration

according to different conditions. Figure 3 shows an example where the dynamic configuration management layer binds the six logical nodes requested by JaMes to six physical nodes provided by the sentinel agent and distributed over two different sites.

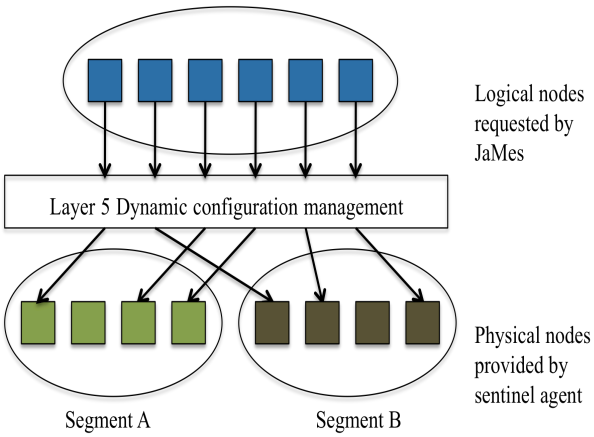

Figure 3 Dynamically mapping physical nodes to logical nodes

3.6 JaMes

JaMes is a distributed parallel programming model that operates within a logical network. It provides basic mechanisms for thread migration and data communication within this network. Each logical node is assigned an arbitrary number (a rank) by the application. Migrating and communication commands use rank numbers to specify their targets. An executing thread can determine its current location by calling JaMes.getRank(), which is a basic function provided by JaMes. Ranks are mapped to physical nodes by the five levels described above when the JaMes application starts.

4. DYNAMIC RESOURCE MANAGER OPERATION

4.1 Bottom-up Resource Supply

Our model supports dynamic resource management using a hybrid top-down/bottom-up approach. In the bottom-up view it constructs a self-organizing network, which supplies a global resource view over multiple segments. Nodes can join and quit the system dynamically as described later in this section.

4.1.1 Information flow within the system

All nodes within a segment need to broadcast their local resource messages to all other nodes within the segment and receive reciprocal information. In addition, each representative node broadcasts information about all nodes in its segment to all other representative nodes and also broadcasts information received from other representative nodes to the other nodes in its segment.

All the above communication is bidirectional, so all nodes maintain resource information about the entire system. This allows the sentinel agent to query only its own local PFAgent, avoiding the potential bottleneck that would arise if all status requests had to be funneled through the representative nodes.

4.1.2 Joining the system

When a new node joins the system, the processing and communication sequence depends on whether the node is part of an existing segment. Figure 4 illustrates the different communication paths.

If the new node is not part of an existing segment, we have to perform the following initial step

a) The node must first register itself with the rendezvous point and establish a tcp-link to all other segments already registered with the rendezvous point. The node becomes the representative node for the system. As part of the registration process, this node receives information about the current status of all nodes on all other registered segments.

If the node is a member of an already-registered segment, or once step a) is complete, we start up the PFAgent on the node, which in turn starts up the UWAgent. The following communication then occurs.

b) The PFAgent on the new node broadcasts its hardware and software information to all the PFAgents on nodes within the same segment. It also receives the reciprocal information from other nodes on the same segment.
c) The representative node for the segment broadcasts the potential field information for all nodes in this segment to the representative nodes of all other registered segments.
d) The representative node on every other segment broadcasts the updated potential field information to all other nodes in its segment.

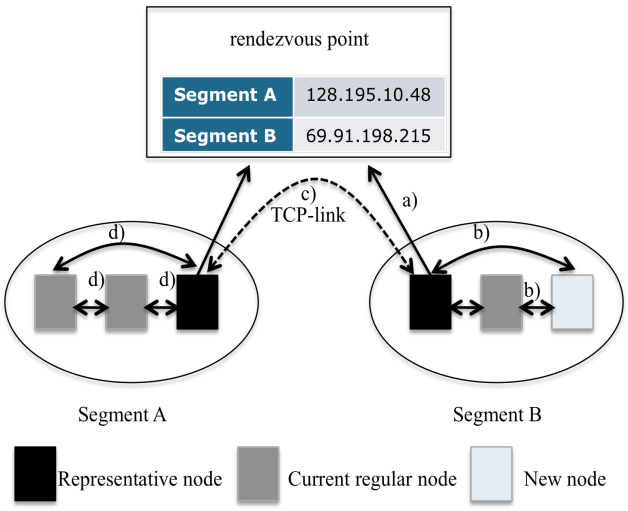

Figure 4 Dynamic nodes management

4.1.3 Quitting the system

If a node crashes or terminates, thus becoming detached from the computing-resource potential field, it

stops broadcasting and hence is removed from the potential field on its segment. Similarly, if the representative node of a segment crashes or terminates all nodes in this segment are removed from the system.

4.2 Top-down Resource Demand

A user can inject an application at any node in the system, using the following command line syntax:

inject app arguments #nodes criteria

This causes the following steps to occur, as illustrated in Figure 5.

a) JaMes calls the configuration manager, passing it the number of nodes and the node selection criteria.
b) The configuration manager spawns the local sentinel agent and passes it the number of nodes and the node selection criteria.
c) The local sentinel agent queries the local PFagent. The PFagent returns information about all nodes on all segments.
d) The local sentinel agent chooses the appropriate number of nodes according to the requested criteria.
e) The local sentinel agent sends a command including the application name and its arguments to all the chosen nodes. There are two situations, depending on the locations of the chosen nodes. For nodes chosen from the same site as the sentinel agent, all inter-process communication is done through sockets. For nodes on remote segments, the interprocess communicating is relayed through the representative nodes on the two segments, which communicate through the TCP link.

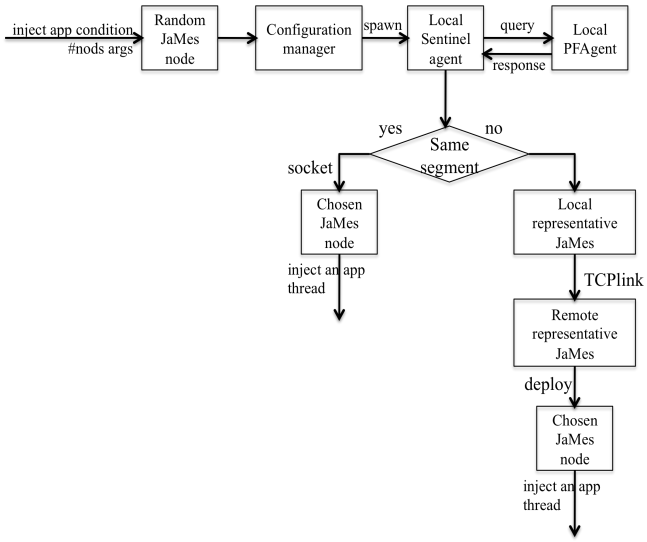

Figure 5 Resource selection by sentinel agent

5. APPLICATION EXAMPLE

In this section a matrix multiplication example is presented to show the usability of our five-layer model. All performance data in this paper was obtained from two campuses: University of California, Irvine (UCI) and University of Washington, Bothell (UWB). The UCI nodes are i686 Intel Pentium 4 CPU's, 3.00GHz, 1GB main memory, and 100Mbps of Ethernet connection. The operating system is Linux version 2.6.22 and the JDK version is JDK 1.6.0. The UWB nodes are Intel Xeon CPU's, 3.2GHz, 512MB main memory, and 1Gbps of Ethernet connection. The operating system is Linux version 2.6.9 and the JDK version is JDK 1.5.0.

5.1 Matrix Multiplication based on JaMes

Matrix multiplication is very important in scientific computing and it is also a classic benchmark for parallel programming. Our solution is based on Gentleman's algorithm [8-9], a classical SPMD algorithm for parallel matrix multiplication. In this algorithm, both matrices A and B migrate as shown by the arrows in Figure 6, while the result matrix C is stationary. In this example we define the grid size to be 3, which divides all matrices into 9 blocks, each held by a separate node. As a result, there are 9 threads on each node, each responsible for the computation of one block.

There are two ways for different nodes to communicate with each other in JaMes. One is by hopping to another node carrying the data, and the other one is shipping data directly to the destination. In this example, we think of the threads as containing blocks of matrix A which they move among nodes as they hop. The data from B is sent using ship statements to be available for the corresponding computation thread. Figure 7 shows the pseudocode for this implementation of Gentleman's algorithm. We can summarize the computation performed by each thread as: (1) receive the B data, (2) compute A*B, (3) ship the B data, (4) hop to the next node. The two statements

```
JaMes.getSurrogate(left node);
x.process(a,step+1)
```

essentially constitute a hop to the left neighbor, carrying A with it. The neighbor computation is based strictly on node number. The actual binding to the physical node is performed at run time by the configuration manager. Similarly, the ship statement causes B to be sent to the node immediately above the current node and the receive statement causes B to be received from the node below the node where the statement is executed. This is specified based on node number, and the binding to physical node is left to the configuration manager.

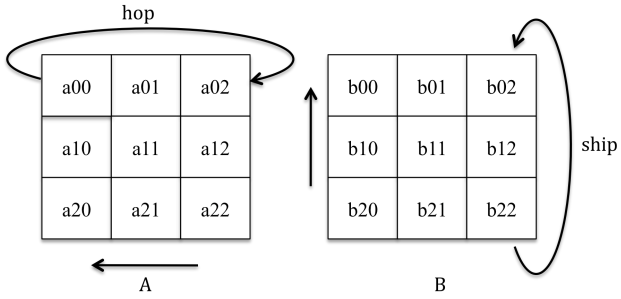

Figure 6. Gentleman's Algorithm.

```
Function process(data, step)
  if (step < gridSize – 1 )
    JaMes.receive(b, down node, ticket)
    C += A * B
    JaMes.ship(b, up node,ticket)
    x = JaMes.getSurrogate(left node)
    x.process(a, step+1)
  end if
end function
```

Figure 7. Pseudocode for JaMes solution.

5.2 Global Resources Discovery and Allocation

According to our model, the representative node is chosen manually for each campus. We choose godzilla on UCI and medusa on UWB as the representatives. Since godzilla and medusa cannot directly establish a TCP connection, we need to create an ssh tunnel. Then for the purposes of this example, we choose 8 nodes, four on each campus, as the resource nodes: hermod0 through hermod3 on UCI and mnode10 through mnode13 on UWB. These nodes collectively form the potential field from which the sentinel agent can make its selection. After all information has been exchanged among the PFAgents at the eight nodes, each PFAgent will contain the same global resource information, which is shown in Figure 8.

PFagent.commandReceiver(): 'show' received from (medusa.uwb.edu)	
UCI campus	UWB campus
###hermod0.ics.uci.edu### disk = 5364 cores = 1 users = 1 os = Linux arch = i686 memory_free = 101 cpu_load = 0.00 processes = 50 cpu_speed = 2994.975 cpus = 1 memory_total = 1010	###mnode10.uwb.edu### disk = 927 cores = 1 users = 1 os = Linux arch = i686 memory_free = 12 cpu_load = 0.00 processes = 56 cpu_speed = 3200.682 cpus = 1 memory_total = 503
###hermod1.ics.uci.edu### disk = 5365 cores = 1 users = 2 os = Linux arch = i686 memory_free = 19 cpu_load = 0.00 processes = 56 cpu_speed = 2994.988 cpus = 1 memory_total = 1010	###mnode11.uwb.edu### disk = 927 cores = 1 users = 1 os = Linux arch = i686 memory_free = 13 cpu_load = 0.00 processes = 56 cpu_speed = 3200.537 cpus = 1 memory_total = 503
###hermod2.ics.uci.edu### disk = 5364 cores = 1 users = 1 os = Linux arch = i686 memory_free = 127 cpu_load = 0.00 processes = 53 cpu_speed = 2995.062 cpus = 1 memory_total = 1010	###mnode12.uwb.edu### disk = 927 cores = 1 users = 1 os = Linux arch = i686 memory_free = 11 cpu_load = 0.00 processes = 59 cpu_speed = 3200.673 cpus = 1 memory_total = 503
###hermod3.ics.uci.edu### disk = 5364 cores = 1 users = 2 os = Linux arch = i686 memory_free = 57 cpu_load = 0.00 processes = 56 cpu_speed = 2994.995 cpus = 1 memory_total = 1010	###mnode13.uwb.edu### disk = 927 cores = 1 users = 1 os = Linux arch = i686 memory_free = 69 cpu_load = 0.16 processes = 56 cpu_speed = 3200.546 cpus = 1 memory_total = 503

Figure 8 Resource information on UCI and UWB campus

Table 1 shows, for different combinations of selection criteria, the physical nodes that will be chosen. For example, specifying CPU speed as the criterion and UCI as the location, the system would choose hermod2, hermod3, and hermod1 to be bound to logical nodes 0, 1, and 2, respectively. The reason for this selection can be seen in Figure 6, where the three chosen nodes are the nodes at UCI with the highest CPU speed. If no performance criteria are specified the system nondeterministically chooses a collection of physical nodes at the requested location based on the order of arrival of UDP messages. One such possible choice is binding hermod0, hermod2, and hermod1 to logical nodes 0, 1, and 2 as shown in the table in the column labeled "random." After the sentinel agent returns the selection, the configuration manager binds these physical nodes to logical rank numbers.

Table 1 Resource table on UCI and UWB

Candidate	hermod0 – hermod3, mnode10 – mnode13			
Condition	cpu_speed	memory_free	random	logical rank
UCI	hermod2	hermod2	hermod0	0
	hermod3	hermod0	hermod2	1
	hermod1	hermod3	hermod1	2
UWB	mnode10	mnode13	mnode10	0
	mnode12	mnode11	mnode12	1
	mnode13	mnode10	mnode13	2

6. RELATED WORK

One focus of this paper is a hybrid mechanism for finding and allocating the most suitable computational resources for an application in a dynamically changing environment. Previous research most closely related to this problem is computing-resource search. Existing literature discusses and

classifies two approaches: broker-based search and brokerless search [10].

The best-known grid-computing middleware systems use broker-base search models [11-12]. In these systems, all computing nodes register their resources with a central broker, and the broker sends to the requesting process the computing resources best suited to to its job requirements. In contrast, in the brokerless-type search model the requester multicasts a resource-search query to servers in its vicinity. These requests may be relayed by the servers to their neighbors, thus implementing a flooding algorithm. This process repeats until suitable resources are found and returned to the requester.

Our hybrid resource management approach can be viewed as a brokerless-type search model. The task performed by the sentinel agent is roughly analogous to spawning to a resource-search query. The crucial difference is that our queries do not need to be propagated through the network. Instead, the potential field agents provide the necessary information, which they maintain up-to-date. In essence, they are serving as a cache for global available resource information.

7. CONCLUSION

We have described a method for dynamically mapping logical nodes in JaMes applications to physical nodes. After starting JaMes on each node, a full TCP connection is set up among all processors with their own UDP and TCP addresses. One of the processors is explicitly selected as the master to collect IP name/port pairs and distribute them to all the other processors. Then an application thread can be injected on the logical node with rank 0.

JaMes applications specify all destinations of computation, migration, and communication in the code, using only rank numbers The rank number for each processor is stationary and unchangeable. Programmers cannot determine the physical destination of migration or communication. The five execution layers can choose the best available hardware and software for the given application. In the bottom-up view layers 0 through 3 construct a self-organizing multi-campus network, which the participated node can join in and quit dynamically. Layers 4 through 6 provide a top-down dynamic mapping mechanism, through which the system can map physical nodes to logical nodes according to different selection criteria.

One advantage of our approach is the flexibility that allows the application to specify its needs in hardware-independent manner using performance and location constraints. Another advantage is the speed with which resource requests can be satisfied. This is due to the monitoring and caching of global performance data carried out by the potential field agents at layer 3 of our model.

In this paper we have only partially addressed the issue of fault-tolerance. In Section 4 we described how nodes can enter and leave the system and how these changes are handled by the PFAgents layer in a transparent manner. As a result the bottom-up portion of our hybrid model is fault-tolerant. However, if a node used by a currently running JaMes program crashes, at present the only alternative is to restart the application. Providing a more graceful recovery scheme for the top down portion of our model, and hence for the JaMes system, is part of our future work.

REFERENCES

[1]. Task Force on Autonomous and Autonomic Systems. First ieee international zconference on self- adaptive and self-organizing systems: Home page http://www.labunix.uqam.ca/ jpmf/saso2007/, 2007.

[2]. Stephan Dudler and Theus Hossmann. Design and implementation of an intra-compartment routing scheme. Fp6-ist-28489, ana deliverable 2.9, ETHZ, September 2008.

[3]. Q. Shang, M. Fukuda, M. B. Dillencourt, L. F. Bic, JaMes: A Java-based System for Navigational Programming. In preparation.

[4]. Ghazi Bouabene, Christophe Jelger, and Ariane Keller. Ana core documentation. Fp6-ist- 27489/wp1/d.1.10, University of Basel, June 2008.

[5]. L. Pan, M. Lai, K. Noguchi, J. J. Huseynov, L. F. Bic, M. B. Dillencourt, Distributed Parallel Computing Using Navigational Programming, International Journal of Parallel Programming, Vol. 32, No. 1, Feb. 2004

[6]. L. Pan, L. F. Bic, and M. B. Dillencourt, Distributed Sequential Computing Using Mobile Code: Moving Computation to Data, L. M. Ni and M. Valero (eds.), Proceedings of the 2001 International Conference on Parallel Processing (ICPP 2001), IEEE Computer Society, Los Alamitos, California, pp. 77–84 (September 2001).

[7]. Munehiro Fukuda, Duncan Smith, "UWAgents: A Mobile Agent System Optimized for Grid Computing", In Proc. of the 2006 International Conference on Grid Computing and Applications, Las Vegas, NV, pages 107-113, June 26-29, 2006.

[8]. W. M. Gentleman. Some complexity results for matrix computations on parallel computers. Journal of the ACM, 25(1):112–115, Jan. 1978.

[9]. L. Pan, W. Zhang, A. Asuncion, M. Lai, M. B. Dillencourt, L. F. Bic, Incremental Parallelization Using Navigational Programming: A Case Study, International Conference on Parallel Processing (ICPP-2005), Oslo, Norway, June 2005.

[10]. Tsutomu Inaba, Takuro Okawa, Yoshitomo Murata, Hiroyuki Takizawa, and Hiroaki Kobayashi. Design and implementation of an efficient search mechanism based on the hybrid p2p model for ubiquitous computing systems. In Proc. of the 2006 Symposium on Applications and the Internet (SAINT'06), pages 45–53, Phoenix, AZ, January 2006.

[11]. Ian Foster and Carl Kesselman. The Globus Project: A Status Report. In Proc. IPPS/SPDP'98 Heterogeneous Computing Workshop, pages 4–18, 1998.

[12]. Andrew Grimshaw, Adam Ferrari, Frederick Knabe, and Marty Humphrey. Wide-area computing: Resource sharing on a large scale. IEEE Computer, Vol.32(No.5):29–37, May 1999.