# Network Sensitive Reconfiguration of Distributed Applications

Kaoutar El Maghraoui, Travis J. Desell, and Carlos A. Varela
*Department of Computer Science*
*Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180-3590, USA*
*{elmagk,deselt,cvarela}@cs.rpi.edu*

## Abstract

*Large-scale, dynamic, and heterogeneous networks of computational resources promise to provide high performance and scalability to computationally intensive applications, but these environments also introduce the need for complex resource management strategies. This paper introduces actor-based programming abstractions and a middleware framework to relieve developers from considering non-functional concerns while allowing middleware layers to optimize application performance and global resource utilization. The Internet Operating System (IOS) consists of a peer-to-peer virtual network of middleware agents that trigger application component reconfiguration based on changes in the underlying physical network and based on the application communication patterns and resource consumption. IOS middleware agents are highly customizable to account for different resource profiling, load balancing, and peer-to-peer interconnection policies.*

*Despite the lack of global coordination and information management, IOS exhibited the ability to reconfigure distributed applications effectively improving their performance over highly dynamic networks. Diverse application communication topologies were tested on Internet-like and Grid-like environments using two middleware agent interconnection topologies: peer-to-peer (p2p) and cluster-to-cluster (c2c). In most cases, p2p agent topologies outperformed c2c agent topologies for Internet-like environments; while c2c agent topologies outperformed p2p agent topologies for Grid-like environments. Our empirical results show also that using group migration of application components to perform load balancing outperforms single migration for the four studied application topologies. These empirical results suggest that adaptive middleware is needed to dynamically change the virtual network topology based on application-level communication patterns and network-level interconnectivity to improve distributed applications performance.*

## 1 Introduction

Viewing millions of computers and devices connected to the Internet as a *World-Wide Computer* enables computationally intensive applications to use otherwise idle distributed resources. One challenge in making this *worldwide computing* vision [27] a reality is to manage resources in a coordinated and efficient way, given the dynamic, heterogeneous, and large scale nature of the network. Another challenge is to provide high-level programming abstractions that facilitate applications development [28].

We present a software framework that enables the development of dynamically reconfigurable distributed applications that autonomously adapt to changes in their execution environment. A middleware layer analyzes both the underlying physical network resources and the application communication patterns to decide how applications should be reconfigured to accomplish load balancing and other non-functional concerns such as fault-tolerance. Resource profiling and reconfiguration decisions are embodied into middleware agents whose behavior can be dynamically modified to implement different reconfiguration policies.

The middleware agents form a *virtual network*. When new nodes join the network or existing nodes become idle, their corresponding agents contact peers to steal work (following Cilk's approach [5]). In previous work [7], the authors showed that considering the application topology in the load balancing decision procedures dramatically improves throughput over purely random work stealing.

In this paper, it is shown that taking the physical network topology into consideration enables the middleware to perform even more effective load balancing over heterogeneous and dynamic networks. Furthermore, the topology of the virtual network also has an impact on the performance and effectiveness of different load balancing strategies. A *peer-to-peer* virtual network topology considers every node in the network the same as every other, while a *cluster-to-cluster* virtual network topology groups nodes according to the perceived distance (as measured by latency) into clusters and elects a manager to perform intra-cluster load balancing. These managers are then connected in a peer-to-peer fashion to perform inter-cluster load balancing. We also evaluate two representative distributed execution environments: an *Internet-like* environment contains nodes with highly diverse processing power and interconnectivity characteristics, while a *Grid-like* environment contains a set of relatively homogeneous clusters with more uniform interconnectivity.

The remainder of the paper is organized as follows. Section 2 describes our model for autonomous worldwide computing. Section 3 describes the architecture of the Internet Operating System (IOS) including modular components for information sharing and decision making by middleware agents. Section 4 describes different load balancing and component migration strategies. Our empirical tests and results are explained in Section 5. The paper concludes with a survey of related work in Section 6 and a discussion in Section 7.

## 2 A Model for Autonomous Worldwide Computing

The development of distributed applications which are efficient, secure, fault tolerant and scalable is a challenging task. Advanced programming models and abstractions are required to alleviate application developers from the burden of resource management and application reconfiguration by delegating these issues to middleware. Middleware can reconfigure applications dynamically as the network and application needs change [4]. The World-Wide Computer (WWC) framework uses high-level programming abstractions that simplify development of applications and enable the middleware to dynamically reconfigure application components to improve performance.

### 2.1 Programming Abstractions

The WWC uses programming abstractions based on the actor model of computation [1, 9]. *Actors* encapsulate memory and communicate via asynchronous message passing. *Universal actors* extend actors by assigning them universal names which enable mobility. *Autonomous actors* extend universal actors to support autonomous (middleware-triggered) reconfiguration.

#### 2.1.1 Actors

The Actor model of computation encapsulates state and process into a single reactive unit of concurrency. Each actor has a unique name, which can be used as a reference by other actors. Communication between actors is purely asynchronous. The actor model guarantees message delivery and fair scheduling of computation. Actors process

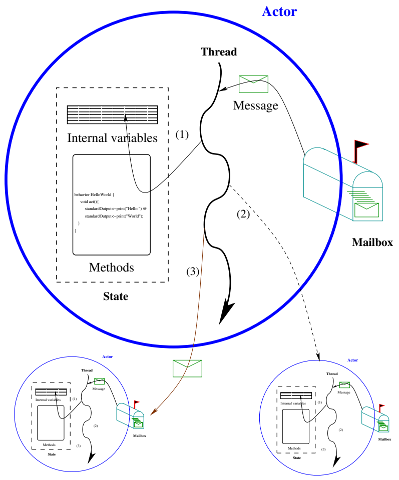

**Figure 1.** Actors are reactive entities. In response to a message, an actor can (1) change its internal state, (2) create new actors, and/or (3) send messages to peer actors.

information in reaction to messages. While processing a message, an actor can carry out any of three basic operations: alter its state, create new actors, or send messages to peer actors (see Figure 1). Actors are therefore inherently independent, concurrent and autonomous which enables efficiency in parallel execution [12] and facilitates mobility [3].

The actor model and languages provide a very useful framework for understanding and developing open distributed systems. Among other applications, actor systems have been used for enterprise integration [25], real-time programming [19], fault-tolerance [2], and distributed artificial intelligence [8].

The distributed memory and asynchronous communication model makes actors ideal for autonomous system reconfiguration. Without any shared memory or blocking communication behavior, actors can migrate in a distributed system without changing the applications semantics.

#### 2.1.2 Universal Actors

In considering mobile computation, it becomes useful to not only model the interactions of actors with each other, but also to model the interactions of actors with their environments. In the actor model, locations are not explicitly represented, therefore semantically there is no difference if two actors are in the same memory space, or on two computers on opposite ends of the earth. However, when considering the problems associated with worldwide computing, it becomes important to represent the actor's environment; to account for different latencies, unreliable environments or heterogeneous resources.

Universal actors are an extension to the actor model providing translucent knowledge of locations (also known as *theaters*), mobility, and the concept of universal names and universal locators. Names represent actor references that do not change with actor migration. Locators represent references that enable communication with universal actors at a specific location. Each location represents an actor's run-time environment and serves as an encapsulation unit for local resources. Ubiquitous resources, such as processing power, have a generic representation—actor references to ubiquitous resources get updated to the local resources upon migration to new locations.

### 2.1.3 Autonomous Actors

When a system is composed of mobile actors, it can be reconfigured arbitrarily, as long as all its utilized resources are ubiquitous [21]. Autonomous actors extend universal actors by: 1) profiling resource consumption, 2) migrating autonomously, 3) splitting and merging to improve scalability, and 4) replicating for fault tolerance. [1]

**Resource Profiling** In determining how much information is profiled, there is a tradeoff between how accurate reconfigurations are and how much overhead is incurred by profiling. Autonomous actors can adopt different profiling strategies to reduce run-time overhead or to collect more information on resource usage. Generally, actors profile processing power, memory, storage, latency and bandwidth. Each actor keeps a record of the number of messages received, messages sent, and messages processed. Autonomous actors can also profile where messages are sent to and received from, as well as the time taken to process or send a message. Based on the profiled information, the middleware's *decision component* (described in Section 3.2) decides how the autonomous actors are to be distributed.

**Autonomous Migration** System reconfiguration via actor migration is triggered when one of these events occur: an actor processes an application or middleware level migration message, or a soft failure occurs.

## 2.2 The World-Wide Computer (WWC)

The World-Wide Computer (WWC) consists of three layers (see Figure 2):

- An *application layer*, consisting of actors which form a graph according to their references to other actors and the corresponding communication frequency.

[1]Split and merge behavior requires application-dependent methods for decomposing and recomposing actors. Replication requires stateless (purely functional) behaviors or immutable state. We only report here on the first two of these extensions.

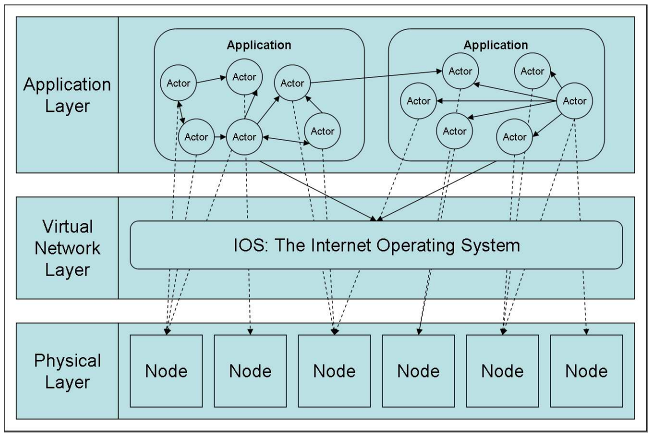

**Figure 2.** A Model for Worldwide Computing. Applications run on a virtual network (the Internet Operating System) which maps actors to locations in the physical layer (the hardware).

- A *physical layer*, consisting of nodes providing computational resources, such as memory, processing power and storage; and connected by physical links providing different latencies and bandwidths.

- A *virtual network layer* between the application and physical layers, consisting of the Internet Operating System (IOS) middleware. It profiles information from the application layer regarding actor connectivity and communication frequency, and from the underlying physical layer regarding node resource usage and availability, and link status, latency and bandwidth. The middleware autonomously reconfigures the application layer components providing non-functional services such as load balancing, fault-tolerance and resource management.

## 3 The Internet Operating System (IOS)

The IOS architecture is decentralized for robustness, scalability and efficiency. Each node at the physical layer contains a *middleware agent*. Based on customizable protocols, agents arrange themselves in various virtual network topologies to trigger application reconfiguration as needed.

## 3.1 Virtual Network Topologies

Virtual network topologies which adjust themselves according to the underlying physical layer are said to be *network sensitive*. We present two types of representative topologies: a *peer-to-peer* (p2p) topology and a *cluster-to-cluster* (c2c) topology. The p2p topology consists of several

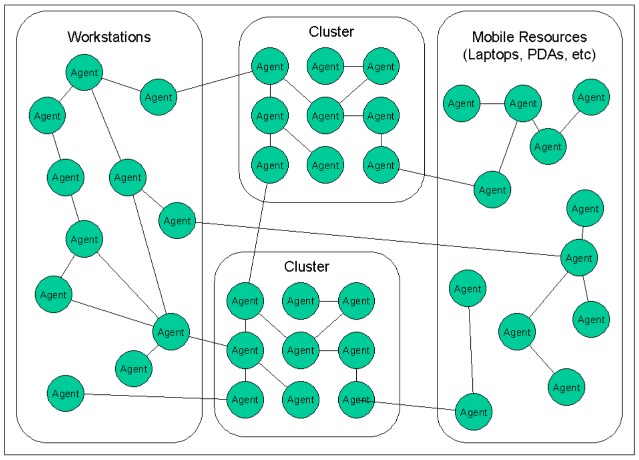

**Figure 3.** The peer-to-peer virtual network topology. Middleware agents represent heterogeneous nodes, and communicates with groups or peer agents. Information is propagated through the virtual network via these communication links.

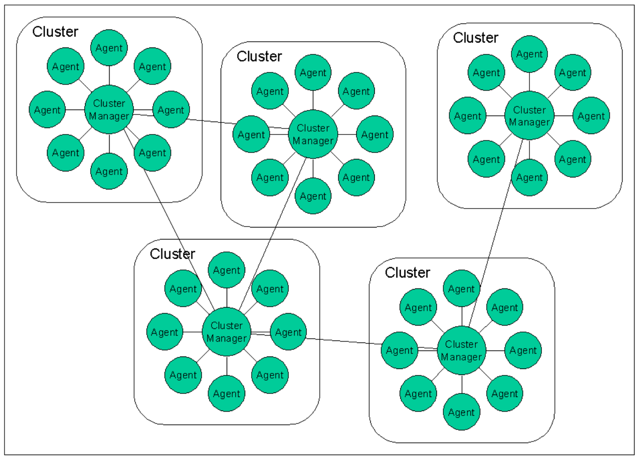

**Figure 4.** The cluster-to-cluster virtual network topology. Homogeneous agents elect a cluster manager to perform intra and inter cluster load balancing. Clusters are dynamically created and readjusted as agents join and leave the virtual network.

heterogeneous nodes inter-connected in a peer-to-peer fashion while the c2c topology imposes more structure on the virtual network by grouping homogeneous nodes with low inter-network latencies into clusters.

**A Network Sensitive Peer-to-Peer Topology (NSp2p)** Agents initially connect to the IOS virtual network either through other known agents or through a *peer server*. Peer servers act as registries for agent discovery. Upon contacting a peer server, an agent registers itself and receives a list of other agents (peers) in the virtual network. Peer servers simply aid in discovering peers in a virtual network and are not a single point of failure. They operate similarly to gnutella-hosts in Gnutella peer-to-peer networks [6]. After an agent has connected to the virtual network, it can discover new peers as information gets passed across peers. Agents can also dynamically leave the virtual network. Previous work discusses dynamic addition and removal of nodes in the IOS middleware [7].

**A Network Sensitive Cluster-to-Cluster Topology (NSc2c)** In NSc2c, agents are organized into groups of virtual clusters (VCs), as shown in Figure 4). Each VC elects one agent to act as the cluster manager. VCs may reconfigure themselves as necessary by splitting or merging depending on the overall performance of the running applications. Cluster managers view each other as peers and organize themselves as a NSp2p virtual network topology.

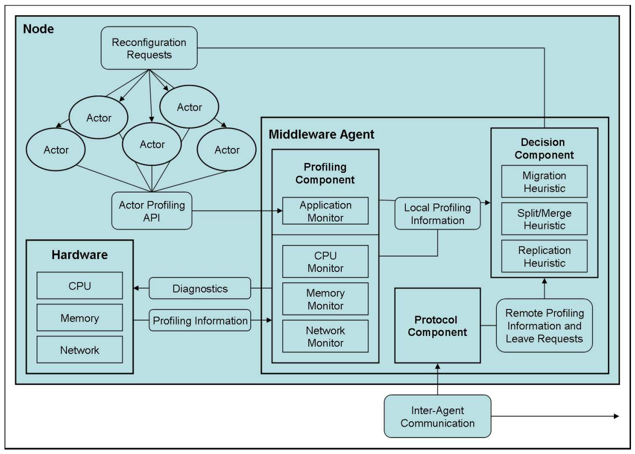

**Figure 5.** Architecture of a node in the Internet operating system middleware. An agent collects profiling information and makes decisions on how to reconfigure the application based on its decision, protocol and profiling components.

## 3.2 Agent Software Architecture

Each node has a middleware agent consisting of three pluggable components (see Figure 5). These components perform the tasks of application and physical layer profiling, inter-agent communication, and virtual network creation. They are also responsible for application reconfiguration decisions. These pluggable components make the middleware highly customizable, and not restricted to any specific method of application reconfiguration or to any particular virtual topology. As such it provides a testbed for comparing multiple methodologies for distributed load balancing, fault-tolerance and other types of reconfiguration.

The agent's components are:

- A *protocol component* to allow for inter-agent communication and virtual network creation.
- A *profiling component* consisting of an application monitor which receives profiled information from actors and different physical resource monitors for profiling processing power, memory and network usage. The architecture modularity affords using different monitoring services, such as the Network Weather Service [30].
- A *decision component* providing models to decide when to migrate, split, merge or replicate actors, given remote profiling information received from the protocol component and local profiling information received from the profiling component. Different models are described in detail in Section 4.

## 4 Autonomous Load Balancing

The implementation of the IOS decision component used in evaluating the different virtual network topologies uses a resource sensitive model (RSM) to balance the resource consumption of actors on the physical layer. RSM provides a normalized measure of the improvement in resource availability an actor would receive by migrating between theaters (see Figure 6 for details). Both the NSp2p and NSc2c protocols use the RSM to decide which actors are the most beneficial to migrate. The following sections describe how this information is used to decide how load balancing is accomplished.

### 4.1 Peer-to-peer Load Balancing

Peer-to-peer load balancing is based on a simple but effective work stealing algorithm described by [5]. Agents configuring themselves in the NSp2p topology keep a list of peers and arrange these peers into four groups based on communication latency [14]: 1) local (0 to 10 ms), 2) regional (11 to 100 ms), 3) national (101 to 250 ms), and 4) global (251 ms and higher).

Agents on nodes which are *lightly loaded* (have more resources available than are currently being utilized) will periodically send reconfiguration request packets (RRPs) containing locally profiled information to a random peer in the local group. The decision component will then decide if it is beneficial to migrate actors to the source of the RRP (the decision making process is described in detail in Section 4. If it decides not to migrate any actors, the RRP is propagated to a local peer of the current agent. This progresses until the RRP's time to live has elapsed, or actors have been migrated. If no actor is migrated, the source of the RRP will send another RRP to a regional peer, and if no migration occurs again, an RRP is sent nationally, then globally. As reconfiguration is only triggered by lightly loaded nodes, no overhead is incurred when the network is fully loaded, and thus this approach is stable[22].

### 4.2 Cluster-to-cluster Load Balancing

The cluster-to-cluster strategy attempts to utilize central coordination within VCs in order to obtain an overall picture of the applications' communication patterns and resource consumption as well as the physical network of the VC. A cluster manager acts as the central coordinator for a VC and utilizes this relatively global information to provide both intra- and inter-VC reconfiguration.

#### 4.2.1 Intra-Cluster Load Balancing

Every cluster manager sends periodic profiling requests to the agents in its respective VC. Every agent responds with information from its profiling component about the local actors and their resource consumption. The cluster manager uses this information to determine which actors should be migrated from the node with the least available resources to the node with the most available resources. Let $n_1$ and $n_2$ be the number of actors running on two nodes, and $r_{i,j}$ be the availability of resource $i$ on node $j$ with a resource weight $w_i$. The intra-cluster load balancing continuously attempts to achieve the relative equality of actors on nodes according to their relative resource availability: $\frac{n_1}{n_2} = \frac{\sum w_i r_{i,1}}{\sum w_i r_{i,2}}$.

#### 4.2.2 Inter-Cluster Load Balancing

For inter-cluster load balancing, NSc2c uses the same strategy as peer-to-peer load balancing, except that each cluster manager is seen as a peer in the network. The cluster managers decision component compares the heaviest loaded node to the lightest loaded node at the source of the RRP to determine which actors) to migrate.

### 4.3 Migration Granularity

RSM supports both single migration and group migration of actors. In single migration, the model is applied to determine an estimation of the gain that would be achieved from migrating an actor from one theater to another. In single migration, one actor is migrated at a time in order to let the application and profiling information readjust to the reconfiguration to prevent incorrect profiling information being used. Group migration tries to speed load balancing by

| Notation | Explanation |
|---|---|
| A | A group of actors. |
| $\mathcal{A}_{r,f}$ | The amount of available resource $r$ at node $f$. |
| $\mathcal{U}_{r,l,A}$ | The amount of resource $r$ used by $A$ at node $l$. |
| $R$ | The set of all resources to be considered by the resource sensitive model. |
| $w_r$ | A weight for a given resource $r$, where $\sum w_r = 1$ |
| $\mathcal{C}_{l,f,A}$ | The cost of migrating the set of actors $A$ from $l$ to $f$ |
| $\mathcal{E}_A$ | The average life expectancy of the set of actors $A$, where $0 \leq \frac{\mathcal{C}}{\mathcal{E}} \leq -1$. |
| $\Delta_{r,l,f,A}$ | The overall improvement in performance the system of actors would receive in terms of resource $r$ by migrating the set of actors $A$ from node $l$ to node $f$, where $\Delta_{r,l,f,A}$ is normalized between -1 and 1. $\Delta_{r,l,f,A} = \frac{\mathcal{A}_{r,f} - \mathcal{U}_{r,l,A}}{\mathcal{A}_{r,f} + \mathcal{U}_{r,l,A}}$ |
| $gain(l, f, A)$ | A normalized measure of the overall improvement gained by migrating a set of actors $A$ from local node $l$ to foreign node $f$. $gain(l, f, A) = (\sum_r w_r * \Delta_{r,l,f,A}) - (\frac{\mathcal{C}_{l,f,A}}{\mathcal{E}_A})$ |

**Figure 6.** The resource sensitive model (RSM) used to by the IOS decision component to determine which actors to migrate between nodes.

migrating multiple actors if resources at the destination theater can feasibly support more than a single actor. Single and group migration with NSp2p and NSc2c are both examined in Section 5.

## 5 Performance Results

This section describes the tests used in the evaluation of the NSc2c and NSp2p virtual networks for load balancing, and the results of this evaluation.

### 5.1 Evaluation Testbed

IOS has been prototyped using SALSA and Java language with high-level constructs for remote messaging, universal naming, migration, and coordination. SALSA programs are compiled into Java code, leveraging the existence of virtual machine implementations in multiple heterogeneous platforms and operating systems.

Our performance results were evaluated using two different physical environments to model Internet-like networks and Grid-like networks. The first physical network consists of 20 machines running Solaris and Windows operating systems with different processing power and different latencies to model the heterogeneity of Internet computing environments. The second physical network consists of 5 clusters with different inter-cluster network latencies. Each cluster consists of 5 homogeneous SUN Solaris machines. Machines in different clusters have different processing power.

Four different application topologies were compared, each pertaining to a level of inter-actor communication and representing different connectivity levels. The unconnected application topology models massively parallel applications where actors continuously perform computations without exchanging any messages. The sparse application topology models applications that have a moderate level of communication but have a higher communication to computation ratio. The tree application topology links actors in a tree structure to model a much higher degree of inter-actor communication. Finally, the hypercube application topology provides the highest amount of inter-actor communication modeling a very high communication to computation ratio.

The unconnected, sparse, and tree application topologies consist of 85 actors performing a set of extensive floating-point operations, while the hypercube application topology consists of 16 actors. The resources that have been monitored by the IOS prototype are CPU performance and network communication. The weights of the communications and CPU resources have been statically assigned for each application. The CPU resource weights were assigned the values of 0.2, 0.5, 0.8, and 1.0, while the communication resource weights were assigned the values of 0.8, 0.5, 0.2, and 0.0 for the hypercube, tree, sparse, and unconnected application topologies respectively.

### 5.2 Virtual Network Evaluation

For most application topologies, NSc2c performed better than NSp2p on grid-like environments (see Figures 7, 8, and 9). The results show the central coordination and knowledge of NSc2c allows more accurate reconfiguration with less overhead due to the homogeneous nature of the clusters in the physical layer. NSp2p lacks this central coordination which explains the decreased performance.

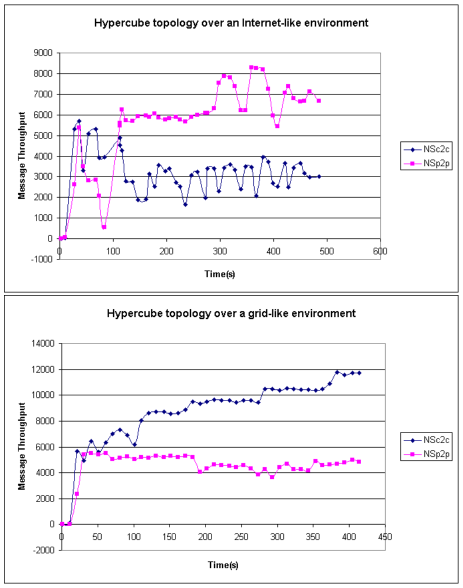

**Figure 7.** The hypercube application topology on Internet- and Grid-like environments.

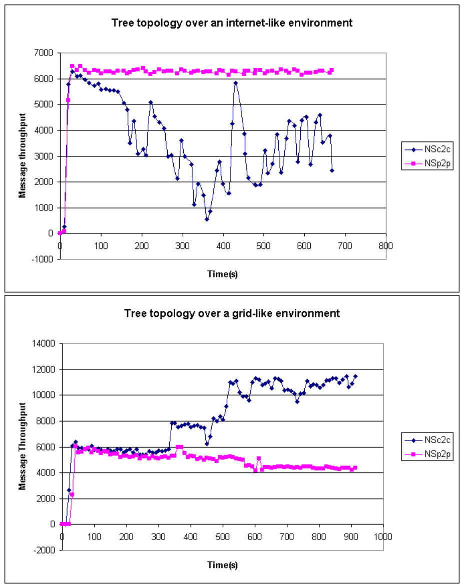

**Figure 8.** The tree application topology on Internet- and Grid-like environments.

In Internet-like environments, NSp2p outperformed NSc2c (see Figures 7, 8, and 10). Due to the lack of homogeneous resources which could be centrally coordinated, NSc2c required additional overhead and performed less accurate reconfiguration. NSp2p did not require the extra overhead of central management and thus proved to be the better strategy on this type of physical network.

However, in both cases NSp2p outperformed NSc2c in the unconnected application topology, as NSc2c outperformed NSp2p in the sparse application topology. This is due to the way the strategies interact with the application topologies. For the unconnected application topology, simply dispersing the actors as much as possible across the network achieves the best load balancing; however with the sparse application topology, more tightly coupled actors should be kept closer together. NSp2p more quickly disperses actors across the physical network, while NSc2c tries to keep actors within the same cluster only migrating actors out when necessary.

## 5.3 Single vs. Group Migration

In the NSc2c strategy, intra-cluster load balancing can possibly migrate multiple actors at the same time given the centralized knowledge at the cluster manager. We have evaluated these two strategies over a testbed with two clusters. Each cluster consists of 4 machines. Results show that for the 4 application topologies, group migration performs better than individual migration (see Figures 11 and 12). While individual migration is more conservative and results in a more stable behavior of the application throughput as the experiments show, migrating multiple actors simultaneously can balance the load much quicker.

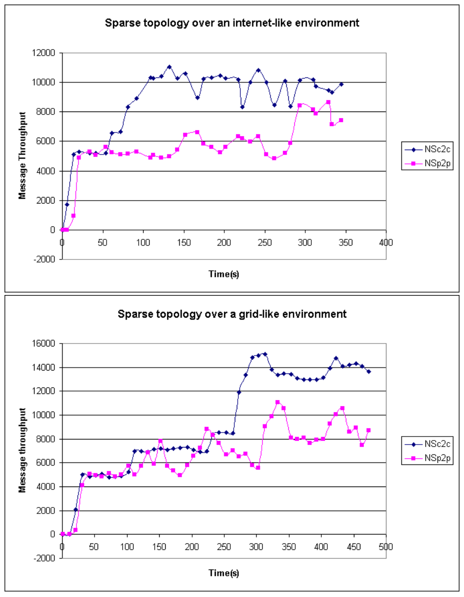

**Figure 9.** The sparse application topology on Internet- and Grid-like environments.

## 6 Related Work

A significant amount of research has been done on load balancing at various system levels. Network-level load balancing tries to optimize the utilization of existing network resources by controlling traffic flow and minimizing the

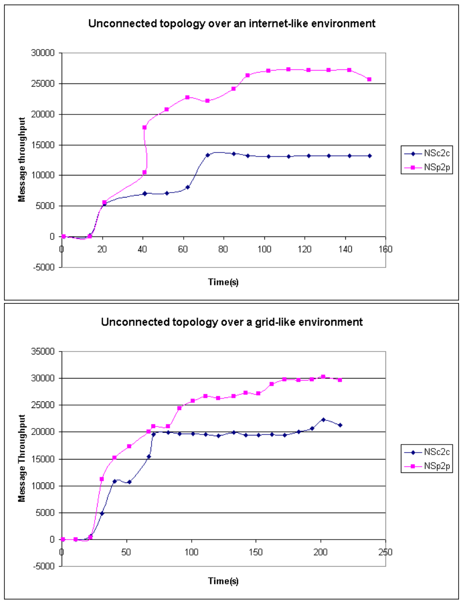

**Figure 10.** The unconnected application topology on Internet- and Grid-like environments.

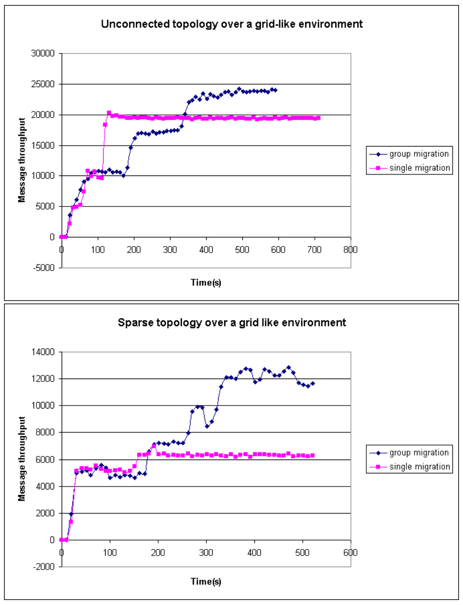

**Figure 11.** Single vs. group migration for the unconnected and sparse application topologies.

number of over-utilized links and under-utilized links [20]. Middleware-based load balancing provides the most flexibility in terms of balancing the load to different types of applications [15]. It is not constrained to the OS or network level but it spans different system levels. Different load balancing strategies exist that range from static to dynamic, centralized to distributed, and sender-initiated to receiver-initiated strategies.

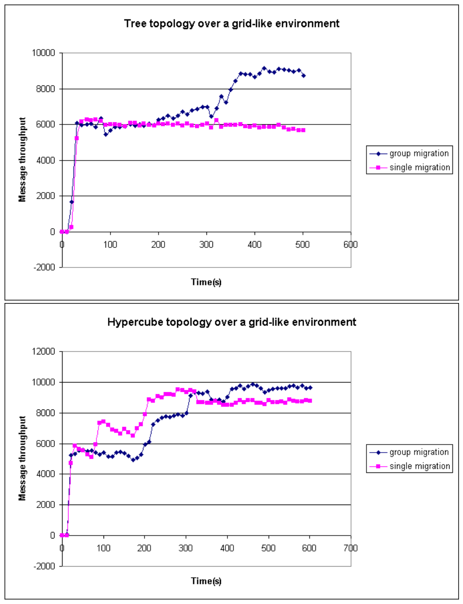

**Figure 12.** Single vs. group migration for the tree and hypercube application topologies.

This work focuses on middleware-level based load balancing strategies across large scale, highly dynamic peer-to-peer networks. Several load balancing strategies have been studied for structured and unstructured P2P systems. Some of them distribute objects across structured P2P systems [18, 24, 10]. They are all based on the concept of distributed hash tables. However they assume that all objects are homogenous and have the same size. Rao et al. [11] have accounted for heterogeneity by using the concept of virtual servers that move from heavy nodes to light nodes which is similar in concept to migration of actors. However they assume that the load on virtual servers is stable. They also assume that there is only one bottleneck resource that needs to be optimized at a time.

Triantafillou et al. [26] have suggested load balancing algorithms to distribute contents over unstructured P2P systems. They aggregate global meta-data over a two-level hierarchy and they use it to re-assign objects. Our load balancing decision functions are not restricted to optimizing a specific bottleneck resource. Our middleware is not restricted to one load balancing strategy. It has been designed and implemented with the intention of plugging in different load balancing strategies depending on the nature of the running applications. This allows us to create concepts and decision functions based on the actor model, where placement of actors can be modified dynamically. We have used this middleware as a testbed to evaluate different strategies with several application communication topologies simulating diverse applications.

The issue of adaptive middleware in distributed systems has been studied by several researchers. Gul Agha et al. have introduced meta-actors to implement different interaction services such as fault tolerance, security, and synchronization [2, 4, 29]. Fabio Kon et al. have presented a model of reflective middleware that allows dynamic inspection and modification of the execution semantics of running applications as a response to changing resources in a distributed environment in order to improve performance [13]. A dynamically adaptable middleware (Comet) for distributed debugging and communication flow synchronization has been proposed by Peschanski et al [16].

Ranganathan et al. have developed a middleware allowing software agents to learn about their context for ubiquitous computing environments [17]. While context-aware agents can learn about their environment to make accurate decisions and take actions based on this information, IOS middleware agents consolidate information at the middleware level and take decisions on behalf of application components.

## 7 Discussion and Future Work

This paper introduced a modular software framework for distributed computing over the Internet. The Internet Operating System (IOS) middleware consists of agents connected in a peer-to-peer topology. These agents trigger dynamic program reconfiguration based on profiled resource usage and availability.

Our preliminary version of IOS [2] has shown that progressively more informed load balancing schemes improve the performance of distributed applications on dynamic networks. It has also empirically enabled us to evaluate different decentralized coordination strategies on diverse application and physical network topologies. The implemented strategies have focused mainly on (1) profiling CPU processing power in distributed physical machines, (2) being sensitive to application actor topologies to attempt colocation of actors with high frequencies of communication [7], and (3) being sensitive to the physical network topologies to attempt minimization of communications over links with high latency and low bandwidth. Since applications have different resource requirements, the ideal level of profiling is highly dependent on the nature of computation that is being performed. For this reason, a modular architecture that enables plugging-in components for different decision, profiling, and coordination strategies has been developed.

Applications with diverse communication patterns were evaluated on two representative physical networks: first, an *Internet/Web computing* environment with highly diverse computation and communication capabilities across its nodes; and second, a *Grid computing* environment with more structured topology: a set of tightly-coupled and homogeneous clusters. Our results show that with applications exhibiting high communication to computation ratios (e.g., hypercube and tree application topologies), the p2p agent topology performs better on Internet-like environments while the c2c agent topology performs better on Grid-like environments. On sparse application communication topologies, c2c outperforms p2p on both Internet- and Grid-like environments, while on unconnected application topologies (representing massively parallel applications), p2p outperforms c2c in both Internet- and Grid-like environments. Using group migration to balance the distribution of load across several nodes performed empirically better than using single migration of actors.

[2] We have used IOS v0.3 for these experiments. The source code is freely available for download at http://www.cs.rpi.edu/wwc/ios/.

The development of the World-Wide Computer is an ongoing process. Results have shown that different middleware profiling, decision and communication strategies perform better for different applications and physical network types. In addition to reconfiguring applications due to changes in application resource consumption and communication patters and the dynamic physical network, the IOS middleware should reconfigure itself—autonomously selecting different profiling and decision strategies depending on its environment. Likewise, the middleware agent topology should adapt to minimize the overhead of inter agent communication and improve the application reconfiguration decision process.

Future work includes: (1) profiling more resources, such as bandwidth, memory, and storage; (2) devising strategies and analytical models for splitting, merging, and replicating components; (3) interoperating with existing high-performance messaging implementations (such as MPI) and evolving standardization efforts in the grid computing community (such as the Open Grid Services Architecture); (4) creating application-independent strategies for fault-tolerance at the middleware level; (5) scaling computations up to thousands of nodes; and (6) providing security mechanisms such as human and software agents authentication and fine-grained resources access control.

Our long term goal is to define, develop, and deploy a platform for worldwide computing that enables resource-intensive applications to locate and allocate resources and adapt to highly dynamic, potentially unreliable, distributed computing environments.

## References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[2] G. Agha, S. Frølund, R. Panwar, and D. Sturman. A linguistic framework for dynamic composition of dependabil-

ity protocols. In *Dependable Computing for Critical Applications III*, pages 345–363. International Federation of Information Processing Societies (IFIP), Elsevier Science Publisher, 1993.

[3] G. Agha, N. Jamali, and C. Varela. Agent Naming and Coordination: Actor Based Models and Infrastructures. In A. Ominici, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents*, chapter 9, pages 225–248. Springer-Verlag, 2001.

[4] G. A. Agha and C. A. Varela. Worldwide computing middleware. In *[23]*. 2004.

[5] R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 356–368, Santa Fe, New Mexico, November 1994.

[6] Clip2.com. The gnutella protocol specification v0.4, 2000.

[7] T. Desell, K. ElMaghraoui, and C. Varela. Load balancing of autonomous actors over dynamic networks. In *Hawaii International Conference on System Sciences, HICSS-37 Software Technology Track*, Hawaii, January 2004.

[8] J. Ferber and J. Briot. Design of a concurrent language for distributed artificial intelligence. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, volume 2, pages 755–762. Institute for New Generation Computer Technology, 1988.

[9] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*, 8-3:323–364, June 1977.

[10] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures*, pages 41–52, Aug. 2002.

[11] A. R. Karthik. Load balancing in structured p2p systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, 2003.

[12] W. Kim and G. Agha. Efficient Support of Location Transparency in Concurrent Object-Oriented Programming Languages. In *Proceedings of Supercomputing'95*, 1995.

[13] F. Kon, F. Costa, G. Blair, and R. H. Campbell. The case of reflective middleware. *Commun. ACM*, 45(6):33–38, 2002.

[14] T. T. Kwan and D. A. Reed. Performance of an infrastructure for worldwide parallel computing. In *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, page 379, San Juan, Puerto Rico, 1999.

[15] O. Othman and D. C. Schmidt. Issues in the Design of Adaptive Middleware Load Balancing. In *Proceedings of the 2001 ACM SIGPLAN workshop on Optimization of middleware and distributed systems*, pages 205–213, Snow Bird, Utah, USA, 2001.

[16] F. Peschanski, J.-P. Briot, and A. Yonezawa. Fine-grained dynamic adaptation of distributed components. In *Middleware 2003*, pages 123–142, Rio de Janeiro, Brazil, June 2003. Springer.

[17] A. Ranganathan and R. H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Middleware 2003*, pages 143–161, Rio de Janeiro, Brazil, June 2003. Springer.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172, 2001.

[19] S. Ren, G. A. Agha, and M. Saito. A modular approach for programming distributed real-time systems. *Journal of Parallel and Distributed Computing*, 36:4–12, 1996.

[20] H. Saito, Y. Miyao, and M. Yoshida. Traffic engineering using multiple multipoint-to-point LSPs. In *INFOCOM (2)*, pages 894–901, 2000.

[21] T. Sekiguchi and A. Yonezawa. A calculus with code mobility. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems, Volume 2*, pages 21–36. Chapman & Hall, 1997.

[22] N. G. Shivratri, P. Kreuger, and M. Ginghal. Load distributing for locally distributed systems. *IEEE Computer*, 25:33–34, December 92.

[23] M. P. Singh, editor. *Practical Handbook of Internet Computing*. Chapman Hall & CRC Press, Baton Rouge, 2004.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, pages 149–160, San Diego, California, August 2001.

[25] C. Tomlinson, P. Cannata, G. Meredith, and D. Woelk. The extensible services switch in Carnot. *IEEE Parallel and Distributed Technology*, 1(2):16–20, May 1993.

[26] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards high performance peer-to-peer content and resource sharing systems. In *First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, pages 341–355, Pacific Grove, California, USA, 2003.

[27] C. Varela. *Worldwide Computing with Universal Actors: Linguistic Abstractions for Naming, Migration, and Coordination*. PhD thesis, U. of Illinois at Urbana-Champaign, April 2001.

[28] C. Varela and G. Agha. Programming dynamically reconfigurable open systems with SALSA. *ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings*, 36(12):20–34, Dec. 2001. http://www.cs.rpi.edu/˜cvarela/oopsla2001.pdf.

[29] N. Venkatasubramanian. Safe composibility of middleware services. *Commun. ACM*, 45(6):49–52, 2002.

[30] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15:757–768, October 1999.