Proc. 5th International Conference on Parallel Processing and Applied Mathematics
R. Wyrzykowski, J. Dongarra, M. Paprzycki and J. Wasniewski (Eds.), Czestochowa,
Poland, September 2003, LNCS, vol. 3019, Springer Verlag, Berlin, pp. 419-430

Dynamically Reconfigurable Scientific Computing on Large-Scale Heterogeneous Grids

Boleslaw Szymanski[1], Carlos Varela[1], John Cummings[2], and Jim Napolitano[2]

[1] Department of Computer Science,
[2] Department of Physics, Applied Physics, and Astronomy
Rensselaer Polytechnic Institute, Troy NY 12180, USA,
{szymansk,cvarela}@cs.rpi.edu, {cummij,napolj}@rpi.edu,
http://www.cs.rpi.edu/

Abstract. Many scientific applications require computational capabilities not easily supported by current computing environments. We propose a scalable computing environment based on *autonomous actors*. In this approach, a wide range of computational resources, ranging from clusters to desktops and laptops, can run an application programmed using actors as program components in an actor language: *SALSA*. SALSA actors have the ability to execute autonomously in dynamically reconfigurable computing environments. We develop the corresponding "Internet Operating system" (*IO*) to address run-time middleware issues such as permanent storage for results produced by actors, inter-actor communication and synchronization, and fault-tolerance in a manner transparent to the end-user. We are using this worldwide computing software infrastructure to solve a long outstanding problem in particle physics: *the missing baryons*, originally identified over thirty years ago.

1 Introduction

Motivated by the needs of many interesting scientific problems, we are developing a solution to finding an inexpensive and abundant computational resource. We intend to target the large collections of laptops, desktops, workstations and clusters intermittently available via the Internet and propose to provide a system that will utilize them efficiently by employing task migration, load balancing [15], and replication at the system level [12]. We also offer users a programming language with convenient and high-level abstractions enabling dynamic system reconfiguration. In addition, we address the issues of portability and security by executing the application within the Java Virtual Machine [11]. To achieve these goals, we are using actors that enable us to separate execution environment issues from application programmer concerns. In that, we differ both from traditional grid computing technologies [6, 9], that do not provide explicit programming and system support for dynamic reconfiguration –therefore, being restricted to mostly static networks of clusters– and from traditional worldwide computing applications (e.g. [13, 7]) that target embarrassingly parallel computation, usually with a single application support.

2 Autonomous Agents over Adaptive Grids

Scientifi c computing is turning to standard computer networks in its search for high performance computing resources, for their price-performance ratio, and their self-upgrading nature. The main challenge of this approach arises from the fact that computer networks are very dynamic and heterogeneous entities with constant node additions, failures, and changes in communication topology. Therefore, there is a need for programming paradigms, models, languages, and software systems that facilitate building dynamically reconfi gurable high-performance distributed systems.

2.1 SALSA platform.

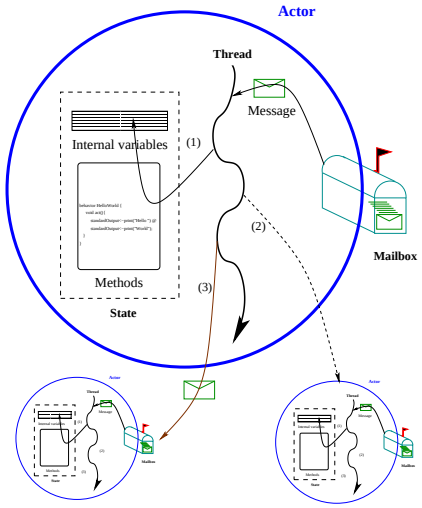


Fig. 1. The actor model of computation

SALSA (Simple Actor Language, System, and Architecture) [15] is an actor programming language (see Figure 1) with high-level constructs for remote messaging, universal naming, migration, and coordination. An actor [1] is a unit of encapsulation for both a state (procedures and data) as well as processing of such a state (a thread of control). All communication between actors is through asynchronous message passing. While processing a message, an actor can carry out any of three basic operations: (1) alter its state, (2) create new actors, or (3) send messages to peer actors. Actors are therefore inherently independent, concurrent, and autonomous, which enables effi ciency in parallel execution and facilitates mobility [2].

SALSA programs are compiled into Java code [8], leveraging the existence of virtual machine implementations in multiple heterogeneous platforms and operating sys-

tems. We therefore view a heterogeneous network of physical machines as a homogeneous network of Java virtual machines. While Java's main perceived drawback is its lack of performance –due to its bytecode interpretation overhead– recent advances in just-in-time (JIT) and adaptive compilation, make Java a very attractive platform for scientific applications [4].

The World-Wide Computer (WWC) [2] run-time architecture consists of naming servers and virtual machines running as Java applications on different Internet nodes. The virtual machines, called *theaters*, provide an environment for execution of universal actors using local resources. High-level programming language abstractions enable actors to create remote communication links with peer actors running on other WWC theaters. Furthermore, actors can easily migrate with their full state to other WWC theaters as they become available, supporting load balancing and scalability. The naming servers keep track of universal actor locators, so that communication remains transparent to actor location and migration.

2.2 Autonomous actors.

We create an extension to actors, *autonomous actors*, that dynamically and opportunistically migrate to nearby theaters, making use of profiled information about their computing and communication patterns [5]. The goal is to enable autonomous actors to find their optimal execution environment without the need for the application programmers to explicitly embed in their codes procedures for load balancing, fault-tolerance, replication, and coordination algorithms.

When a system is composed of mobile actors, it can be reconfigured arbitrarily, as long as all its used resources are ubiquitous. Autonomous actors extend actors by:

- profiling computing and communication performance,
- including a *satisfaction threshold* to trigger migration, and
- introducing message priorities –to enable, e.g., a migration message to take precedence over other messages.

We develop an "Internet Operating system" (IO) middleware layer to help us evaluate different adaptive data and task migration and replication algorithms. If an actor is processing messages at a speed faster or equal to the speed that it is receiving them, it can maintain a constant mail-queue. This means that the actor's current location has enough system resources to handle its tasks. However, when this condition is not met, the messages in an actor's mailbox begin to accumulate. In this case, the unsatisfied actor attempts to migrate to a more appropriate location. Likewise, if new resources become available in the network, or resources are going to be temporarily unavailable, an autonomous actor can choose to migrate to improve overall system performance.

Figure 2 displays a high-level architecture for the proposed IO system. SALSA programs are at the application layer –a program creates and initiates a set of actors which compute and collect results. IO coordinates the mapping of these application-level actors into actual run-time environments (theaters) at the system layer to support autonomous reconfiguration.

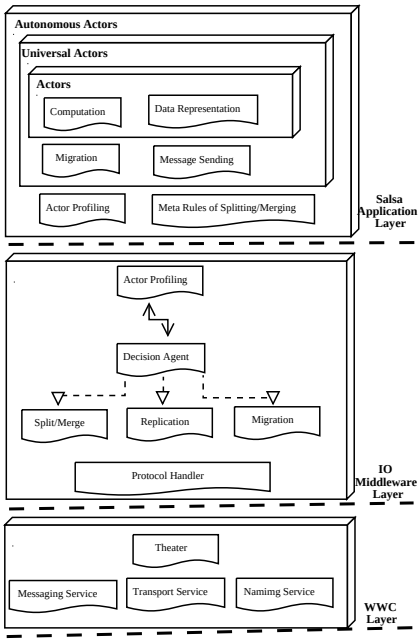


Fig. 2. System Layers

In short, our project implements a middleware system based on actors implemented in SALSA. This middleware supports computations that can be abstracted as a multi-level and dynamic computational tree. In this paradigm the application domain is iteratively divided into subdomains, each creating a branch rooted in the parent domain. Each leaf can either span a subtree or execute its program over its subdomain. Each subdomain computation can be iterated over with synchronization of each iterative step. The communication requirement that allows for effi cient execution in such an environment is of course technology-dependent; however, given the current ratio of communication to computation speed, there is a whole class of important applications that will execute effi ciently in such a scheme.

3 Strategies for Autonomous Load Balancing

In this section, we describe various methodologies for load balancing that vary by the amount of profi ling done and the complexity of the decision agents. The simplest decision agents take into account the load of the individual theaters and autonomous actors, while the more complex agents consider additional factors such the network and actor topologies. All the network protocols are peer-to-peer in nature to allow for maximum scalability. In all cases, a theater joins the autonomous network by registering with a peer server and receiving addresses of other peers in the network from it. Peer servers are not centralized, as many may serve addresses of peers for a single network.

Before describing the strategies for autonomous load balancing, the following concepts are used to describe the attributes of an actor system over a heterogenous network.

Actor Satisfaction Actor satisfaction is a measure of an actor's ability to process and send messages. If an actor is not satisfied, it cannot process messages as quickly as it is receiving them. This includes the cost of message sending, because processing a message also involves sending messages. When an actor is unable to handle the load of messages it is receiving, the size of its message queue begins to increase. An actor with an increasing message queue is unsatisfied.

Theater Load Every theater hosts a group of active actors. A theater is considered lightly loaded if all its actors are satisfied, whereas a theater is considered heavily loaded if at least one of its actors is not satisfied.

3.1 Load-Sensitive Random Stealing (RS)

The simplest strategy is based on random work stealing, a simple but effective algorithm described by [3]. We modified this algorithm to work in a peer-to-peer network by randomly propagating a random steal packet over the network. A lightly loaded theater chooses a neighbor at random and sends it a steal packet. This continues from theater to theater until a candidate for migration is chosen or the packet's time to live has been reached. When either occurs a notification is sent back to the originating theater. This prevents a theater form performing multiple steals simultaneously. One benefit of random steal propagation is that it avoids costly broadcasts to the network, reducing the impact of the middleware on the application. In RS, a peer theater finds its first unsatisfied actor (if one exists) and selects that as its candidate for migration. Also, since only lightly loaded theaters send steal packets, with high loads the overhead for RS becomes almost non-existant.

3.2 Actor Topology Sensitive Random Stealing (ARS)

Actor topology sensitive random stealing builds on the previous strategy by using additional profiling information. Actors monitor the number of messages they send to remote theaters, allowing this strategy to find a actor placement in the network according to the communication patterns in the application. This approach enables actors with high frequencies of communication to be co-located or located in nodes with low latencies, according to the results of a decision function.

The decision function estimates the increase in the performance of an actor if it migrates to a specific foreign theater. Random steal packets now also contain the available processing power of their origin theater. Let $\Delta(l, f, a)$ denote the normalized increase in performance of actor a that results from migrating a from the local theater l to the foreign theater f. The normalized increase in performance is determined by the normalized increase in communication, $\Delta_c(l, f, a)$ and the normalized increase in processing, $\Delta_p(l, f)$ that would occur in that actor after this migration. Migration only happens when the estimated change in throughput is positive ($\Delta(l, f, a) > 0$).

The following equations illustrate how the decision function is evaluated.

$$\Delta(l, f, a) = \Delta_p(l, f) + \Delta_c(l, f, a) \tag{1}$$

where

$$\Delta_p(l, f) = \frac{\mathcal{P}(f) - \mathcal{P}(l)}{\mathcal{P}(f) + \mathcal{P}(l)} \tag{2}$$

and

$$\Delta_c(l, f, a) = \frac{\mathcal{M}(f, a) - \mathcal{M}(l, a)}{\mathcal{M}(a)} \tag{3}$$

We used the following notation: $\mathcal{M}(t, a)$ denotes the number of messages communicated between an actor a and theater t, while $\mathcal{M}(a)$ denotes the total number of messages sent by actor a. $\Delta(l, f, a)$ stands for the normalized change in actor performance obtained by migrating actor a from theater l to theater f, while $\Delta_c(l, f, a)$ and $\Delta_p(l, f)$ denote actor performance change due to communication and message processing, respectively. $\mathcal{P}(t)$ is the processing power available in theater t.

This decision function was chosen because while it is not very precise, it does provide very reasonable results with a minimal amount of overhead. It places a strong emphasis on inter-actor communication and tries to colocate tightly coupled actors (actors which frequently communicate). Both the difference in processing and communication have been normalized to a value between -1 and 1, and in the future we could add weights to these values based on empirical research.

3.3 Network Topology Sensitive Random Stealing (NRS)

In addition to resource availability, NRS takes into consideration the topology of the network. In the IO network a peer might belong to local, regional, national, or international clusters [10]. In these cases, while bandwidth may be high, latency will play a large factor in the throughput of messages between theaters. NRS locates tightly coupled actors close together in the IO network, but allows loosely coupled actors to migrate more freely, as they do not need this restriction.

NRS classifies its neighbors into four groups: local, regional, national and international. These groups are classified into locales by the following ping times [10]:

- Local: 10 ms or less
- Regional: 11 ms to 100 ms
- National: 101 ms to 250 ms
- International: 251 ms and higher

The algorithm then proceeds similar to cluster-aware random stealing described by [14]. Random steal packets specify which locale they are to travel. A theater first selects a local peer randomly and sends a local random steal packet. A theater will only propagate a steal packet to its specified locale. If a local random steal packet fails (the theater receives a terminated packet without an actor), the theater will then attempt a regional random steal, and so on.

Using this method to propagate random steal packets through the network keeps groups of coupled actors close together in the network. NRS uses the previously mentioned methods for determining the best candidate actor when a random steal packet reaches a theater, thus NRS comes in two versions: RS and ARS.

4 Preliminary Results

We ran a series of tests on our IO system using a manual round robin placement of actors (RR), peer-to-peer random stealing (RS) and the actor topology sensitive random stealing (ARS) strategies.

We ran four simulations each pertaining to a level of inter-actor communication. The unconnected actor graph had actors simply process messages over and over, with no inter-actor communication. The sparse actor graph linked actors randomly, providing a moderate amount of inter-actor communication. The tree simulation linked actors in a tree structure, for a higher amount of inter-actor communication. Lastly, the hypercube provided a very high amount of inter-actor communication. (see Figures 3 and 4. We compared throughput of RS and ARS to manual load balancing to measure the overhead that the IO middleware incurred on the computation. All actors were loaded in a round robin fashion across the eight theaters, then were allowed to compute until their throughput leveled off. Throughput is the number of messages processed by all actors in a given amount of time – the higher the throughput, the faster a computation is running.

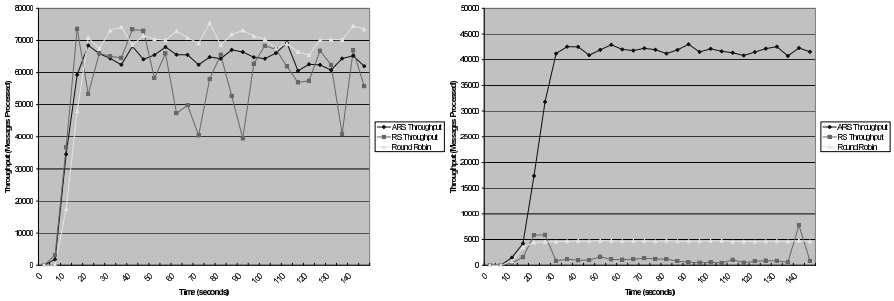


Fig. 3. Unconnected and Sparse Graph Actor Topologies.

Figure 3 shows that both ARS and RS imposed a minimal amount of overhead for the simulation, as a round robin placement of actors is the optimal load balancing solution for an unconnected graph of actors in a homogeneous network, and the round robin placement imposed no middleware overhead. ARS and RS performed comparatively to RR in this test. On the more communication-bound simulations (see Figure 4), ARS outperformed both the manual load balancing and RS. On a sparsely connected graph, ARS performed superbly, bringing throughput to nearly the level of an unconnected graph. In all simulations involving inter-actor communication, ARS highly outperformed RR and RS, showing that the co-location of actors signifi cantly improves message throughput. RS was shown to be too unstable in all these simulations and did not outperform either RR or ARS. Our conjecture is that because the Java thread scheduling mechanism is not fair, actors are found to be unsatisfi ed when they are actually not, leading to the unstable migration behavior of actors when IO uses RS.

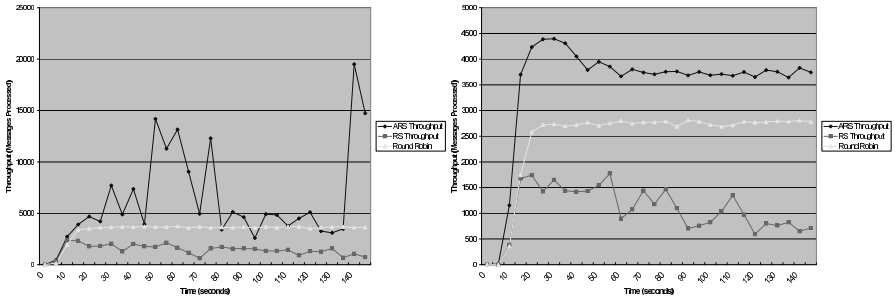


Fig. 4. Tree and Hypercube Actor Topologies.

To show how IO can handle a dynamically changing network, the same simulations were ran on a changing network of peer theaters. The simulations were loaded entirely onto one peer theater, then every 30 seconds an additional peer theater was added to the computation. After eight peer theaters had joined the computation, IO was allowed to run for two minutes to balance the load, after which a peer theater was removed every 30 seconds, until the computation was entirely running on the last peer theater added to the computation.

With the unconnected graph join/leave simulation (see Figure 5), both RS and ARS performed well in distributing the load across the peer theaters (see Figure 6), and increased the throughput by a factor of about six when all eight theaters had joined the simulation. The addition and removal of peer theaters shows that IO can rebalance load with removal and addition of nodes without much overhead.

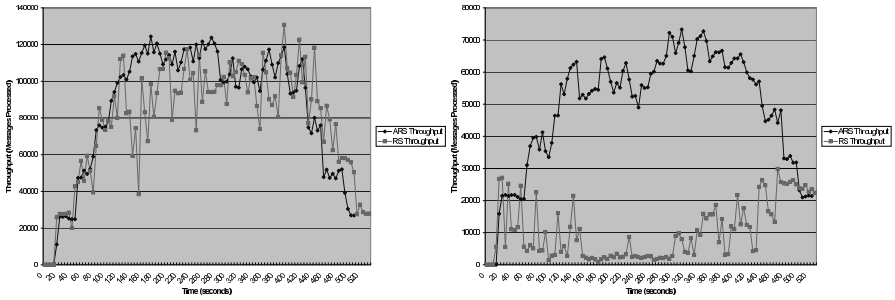


Fig. 5. Unconnected and Tree Graph Actor Topologies on a Dynamic Network.

The graphs of actor placement (see Figure 6) show that while both ARS and RS managed to distribute the actors evenly across the network of theaters, ARS co-located actors more appropriately accorting do their connectivity, signifi cantly improving overall throughput.

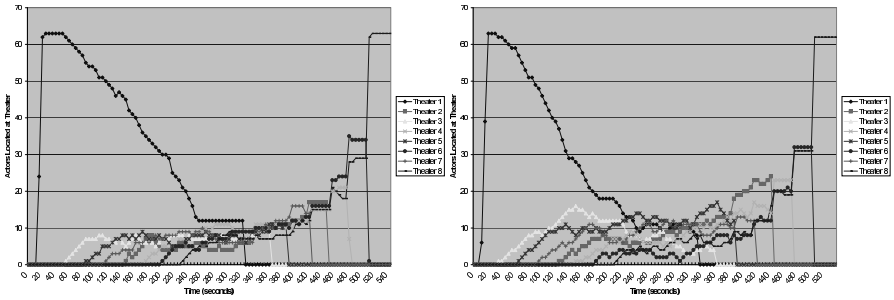


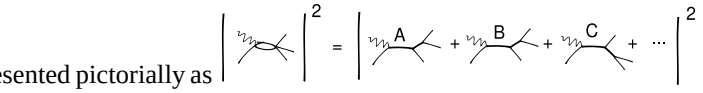
Fig. 6. Actor Distribution for Tree Graph Actor Topology with ARS and RS.

These preliminary results show that the IO system with ARS performs well in most situations for load balancing of a distributed actor system. While the more traditional strategy of random stealing does not fare so well in an autonomous system of actors, a more intelligent strategy can exploit the properties of the actor model to provide autonomic solutions for load balancing across a dynamic network. The results also show that IO can handle the addition and removal of nodes from a computation without any central coordination, a necessity for large dynamic heterogeneous networks.

5 Missing Baryons

Baryons are the family of spin one half states that the well known proton and neutron belong to. In spite of the fact that at present we don't know how to calculate the baryon spectrum with QCD, properties such as symmetry rules can be used to make some general predictions. Baryon states are formed from three quarks bound strongly, and treating the three quarks equally, it is not difficult to write down a list of allowed states in terms of fundamental properties such as the states angular momentum and parity. Interestingly, this list includes quite a few states that have not been observed. Since many of these extra predictions seem to be states that would have been seen by previous and current experiments, they are referred to as "Missing Baryons". The measurement of the baryon spectrum is a extremely helpful, if not necessary, component to understanding Quantum Chromodynamics, QCD, a field theory that describes the strong nuclear force.

The measurement of baryon spectra is the measurement of quantum mechanical states. This means we actually observe a probability distribution, and so to make an accurate measurement we must observe many interactions. To study states produced in a photon-proton interaction, a beam of photons is directed at a target containing many protons. Each photon in the beam has a chance to excite the proton into another state of the baryon spectrum. If such an interaction happens, the produced state very quickly decays into more stable particles which can be seen in a detector. By studying the distribution of these detected particles angles and energies, the quantum mechanical properties of the produced, short-lived, states can be determined.

There are many possibilities for short-lived state, and since the system is described by quantum mechanics, these are described by interfering amplitudes. A technique known as Partial Wave Analysis is used to write down a general expression for the intensity distribution in terms of the various possibilities for this short-lived intermediate state. The amount of the different intermediate states is then varied by a fitting program to match the observed distributions. Thus the intensity distribution, expanded this way, can be represented pictorially as 

where A, B, C, ... represent different intermediate states.

A maximum likelihood fit finds the mix of intermediate states that best describes the data. The usual trick is to minimize the negative log likelihood; for this type of analysis the function looks like:

$$-\ln(\mathcal{L}) \;=\; \sum_{i}^{n} \left| \sum_{\alpha}^{N} \mathrm{V}_{\alpha} \psi_{\alpha}(\tau_i) \right|^2 - n \sum_{\alpha\alpha'} \mathrm{V}_{\alpha}^{*} \Psi_{\alpha\alpha'} \mathrm{V}_{\alpha'} \qquad (4)$$

where α is summed over the N intermediate states, the V's are the fit parameters – the amount of state α required to describe the data. The first sum over i is over the number of interactions observed, n.

Since missing baryon states may be "small" rather than "missing", a detailed search for them requires many interactions. Data samples on the order of $n = 10^6$ are already available to be analyzed, and future experiments expect to increasing this by *two orders of magnitude*. Typically N, the number of intermediate states required, is in the range of 10-100. The numbers in the likelihood calculation are all complex due to the quantum mechanical nature of the problem. So the calculation of the likelihood for a single set of parameters V, requires $n(N+1) + 2N^2$ complex multiplications and nN complex additions. Consider that a minimization may require thousands of evaluations of $-\ln(\mathcal{L})$ in the search for the optimal set of parameters, and the net result is that complete studies are limited by available computational resources. Our model for grid evaluation of this function is essentially the trivial one: the large sum over n data points is split into m smaller sums which are delegated to actors in the World-Wide Computer.

We have recently completed development of a maximum likelihood fitter for partial wave analysis in SALSA and are beginning to verify and performance analyze the package. Some preliminary results of measurements done using an early version which performed χ^2 fits is shown in Figure 7. A large number of data points, 4×10^4, were fit. The time to converge decreases like $1/m+$ constant with increasing m, the number of actors (computers) participating in the computation. The different curves correspond to varying the size of packet transferred between actors, small blocks producing the slow curves on top. These small packet sizes lead to a large overhead due to latency as is demonstrated in the high value of the horizontal asymptote for those curves. Similar timing measurement will be done with the maximum likelihood fits, pushing the tests to much higher number of data points and actors. Each maximum likelihood evaluation requires more computation than the corresponding χ^2 evaluation, and will probably benefit even more from our SALSA based evaluation model.

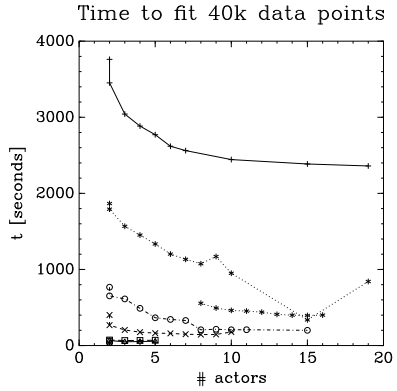


Fig. 7. Timing measurements for χ^2 fits to 4×10^4 data points

6 Conclusion

Our research addresses the following major questions: (i) how to provide a programming environment that allows computing non-specialists to develop new parallel and distributed applications using their programming skills? (ii) how to provide high availability, i.e., adding, removing and reorganization of computers without directly involving application programmers? (iii) how to support parallelism management, i.e., the dynamic management of parallel processes and cluster resources? (iv) how to provide transparency to all programmers of a cluster or network of computing resources and relieve them of fine tuning their applications, which is very time consuming, error prone and largely irrelevant? (v) how to automatically and transparently provide fault tolerance to parallel applications?

The developed software infrastructure[3] provides a modular, extensible, scalable infrastructure to prototype research ideas and new algorithms and methodologies for load balancing, fault-tolerance and coordination in highly reconfigurable distributed systems. Ultimately, we want to provide an efficient middleware infrastructure that enables scientists to develop large-scale high-performance computing applications executing over dynamic grids in a way as transparent as possible to system developers and users.

Acknowledgements

Many ideas presented here are the result of countless discussions in the IO research and development group at RPI. In particular, we would like to express our gratitude to Travis Desell and Kaoutar El Mahgraoui for implementing the ARS and NRS strategies, and to Chris Kona, Heschi Kreinick, and Zack Goldstein for implementing the SALSA fitting programs. The work described here has been supported in part by a Rensselaer Polytechnic Institute Seed Funding Grant.

[3] Source code and documentation available for downloading at http://www.cs.rpi.edu/wwc/io/

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.
2. G. Agha, N. Jamali, and C. Varela. Agent naming and coordination: Actor based models and infrastructures. In A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, chapter 9, pages 225–246. Springer-Verlag, Mar. 2001.
3. 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.
4. J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman. Benchmarking java against c and fortran for scientific applications. In *Proceedings of ACM Java Grande/ISCOPE Conference*, pages 97–105, 2001.
5. T. Desell, K. E. Maghraoui, and C. Varela. Load balancing of autonomous actors over dynamic networks. In *Proceedings of the Adaptive and Evolvable Software Systems: Techniques, Tools, and Applications Minitrack of the Software Technology Track of the Hawaii International Conference on System Sciences (HICSS'37)*, January 2004.
6. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
7. P. Fry, J. Nesheiwat, and B. Szymanski. Computing Twin Primes and Brun's Constant: A Distributed Approach. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 42–49. IEEE Computer Society, July 1998.
8. J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison Wesley, 1996.
9. A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, Jan. 1997.
10. 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*, San Juan, Puerto Rico, 1999.
11. T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison Wesley, 1997.
12. M. Nibhanapudi and B. K. Szymanski. *High Performance Cluster Computing*, volume I of *Architectures and Systems*, chapter BSP-based Adaptive Parallel Processing, pages 702–721. Prentice Hall, New York, 1999.
13. W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on project serendip data and 100,000 personal computers. In *Proceedings of the Fifth International Conference on Bioastronomy*, 1997. Available at http://setiathome.ssl.berkeley.edu/learnmore.html.
14. R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal. Efficient load balancing for wide-area divide-and-conquer applications. *ACM*, 36:34–43, 2001.
15. 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.