Design and Implementation of a Distributed Evolutionary Computing Software

K. C. Tan, *Member, IEEE*, Arthur Tay, and J. Cai

***Abstract*—Although evolutionary algorithm is a powerful optimization tool, its computation cost involved in terms of time and hardware resources increases as the size or complexity of the problem increases. One promising approach to overcome this limitation is to exploit the inherent parallelism of evolutionary algorithms by creating an infrastructure necessary to support distributed evolutionary computing using existing Internet and hardware resources. This paper presents a Java-based distributed evolutionary computing software (Paladin-DEC), which enhances the concurrent processing and performance of evolutionary algorithms by allowing inter-communications of subpopulations among various computers over the Internet. Such a distributed system enables individuals to migrate among multiple subpopulations according to some patterns to induce diversity of elite individuals periodically, in a way that simulates the species evolve in natural environment. The Paladin-DEC software is capable of keeping data integrity throughout the computation, and is incorporated with the features of robustness, security, fault tolerance, and work balancing. The effectiveness and advantages of the Paladin-DEC are illustrated upon two case studies of drug scheduling in cancer chemotherapy and searching probe sets of yeast genome.**

***Index Terms*—Distributed systems, evolutionary algorithms, parallel algorithms.**

I. Introduction

EMULATING the Darwinian–Wallace principle in natural selection and genetics, evolutionary algorithms (EAs) have been found to be very effective in solving complex optimization and machine learning problems [1]–[3]. Unlike traditional single-point gradient-guided search techniques, an evolutionary algorithm intelligently searches the solution space by evaluating performances of multiple candidate solutions simultaneously and approaches the global optimum in a non-deterministic manner. Although evolutionary algorithm is a powerful tool, the computational cost involved in terms of time and hardware resources increases as the size or complexity of the problem increases, since the algorithm often needs to perform a large number of function evaluations during the evolutionary search. Moreover, the EA usually requires a large population and generation size in order to simulate a realistic evolutionary model with good diversity and convergence, which is sometimes cost prohibitive or impractical without the help of high performance computing resources.

Manuscript received August 31, 2003; revised March 25, 2002 and June 17, 2003. This paper was recommended by Guest Editors W. Pedrycz and A. Vasilakos.

The authors are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117576 (e-mail: eletankc@nus.edu.sg).

Digital Object Identifier 10.1109/TSMCC.2003.817359

One promising approach to overcome these limitations is to exploit the inherent parallel nature of evolutionary algorithms by creating an infrastructure necessary to support distributed evolutionary computing for parallel information processing, i.e., to divide a task into subtasks and to solve the subtasks simultaneously using multiple processors. This divide-and-conquer approach has been applied to EAs in different ways and many parallel EA implementations have been reported in literature [4]–[6]. Levine [7] developed the package of PGAPack, which is a parallel evolutionary algorithm library that supports for global parallelization. The package was written in C programming language and the communication can be carried out using MPI which is a popular library specification for message-passing.

Andre and Koza [8] used the transputers hardware and Tomassini and Fernandez [9] applied the MPI as a platform for the implementation of parallel EAs. Tanev *et al.* [10] developed a parallel EA based on distributed component object model (DCOM) which is a protocol that enables software components to communicate directly over the network. Chong [11] proposed an application of Java applet on the Internet that focuses on the massive distributed computing approach, and the distributed resources evolutionary algorithm machine (DREAM) is a project aimed to provide the technology and software infrastructure necessary to support the next generation of evolving infohabitants in a way that makes the infrastructure universal, open and scalable [12].

The availability of powerful networked computers nowadays presents a wealth of computing resources that can provide the processing power required to solve problems unsolvable using a single computer. This potential computational power can be much stronger than a supercomputer, and allows many real and large problems to be tackled by formulating the problems for parallel processing in a distributed computing infrastructure, i.e., to decompose a task into subtasks and to solve these subtasks simultaneously using multiple processors connected over the Internet. Although the distributed computing resources are attractive, the heterogeneous hardware and software available for the Internet have given insurmountable difficulties to the implementation of an open and secured distributed system. However, the recent emergence of Java technology, which is an object-oriented and platform-neutral programming language developed by Sun Microsystems, has provided an opportunity for creating such a distributed system effectively and efficiently.

This paper presents a distributed evolutionary computing software (Paladin-DEC), which enhances the concurrent processing and performance of evolutionary algorithms by allowing inter-communications of subpopulations among multiple computers distributed over the Internet. The Paladin-DEC software incorporates different evolutionary algorithms, such

1094-6977/03$17.00 © 2003 IEEE

as genetic algorithms (GAs), genetic programming (GP), and evolutionary strategies (ES), which have been found to be very effective in solving complex optimization and machine learning problems [13]. The software is developed based upon the technology of Java 2 platform of Enterprise Edition (J2EE), and is an open system for which new codes could be easily added by following the class hierarchy and new problems could be solved by formulating the problems into java files for execution in the system. There are also user-friendly GUIs provided in the software for easy use and configuration, e.g., users can easily select a job from the job queue or set the working parameters through GUIs in the control panel. Besides having the features of reusability, security and work balancing, the Paladin-DEC software is also designed with good robustness and fault tolerance, i.e., single peer failures will have little influence to the system and the server is sufficiently robust to handle general exceptions while the program is running.

This paper is organized as follows. A brief introduction to parallel and distributed evolutionary algorithms is presented in Section II. The design and implementation of the Paladin-DEC software that include a brief description of its various features and GUIs are given in Section III. Section IV illustrates the advantages of the Paladin-DEC software based upon two case studies of drug scheduling in cancer chemotherapy and searching probe sets of yeast genome. Conclusions are drawn in Section V.

II. Parallel and Distributed Evolutionary Algorithms

Evolutionary algorithm is an umbrella term used to describe computer-based problem solving systems which apply computational models of evolution as key elements in the design and implementation. A variety of evolutionary algorithms have been proposed, including evolutionary strategies (ES), evolutionary programming (EP), genetic algorithms (GA), and genetic programming (GP). These algorithms have been developed upon the synthesis of natural evolution which exhibit global search capabilities by simultaneously evaluating performances at multiple points in the solution space. Before this simulated evolution process begins, an initial population of multiple coded chromosomes representing random candidate solutions is formed, and every such chromosome is assigned a performance index. At each generation of search, multiple candidates are evaluated and the search will be directed intelligently according to the Darwin's "survival-of-the-fittest" principle. Then useful search information and coordinates are exchanged and altered for the next generation of candidate solutions. This evolution cycle will be repeated until the final generation is reached or the solution has been found. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful mechanisms for global search and optimization.

Intuitively, the computational workload involved in an evolutionary search process could be extensive since multiple candidate solutions need to be evaluated simultaneously, particularly for difficult optimization problems where a high computational effort is needed for a large population and generation size in order to search for the global optimal solutions. Exploiting the intrinsic parallelism of EAs, various parallel

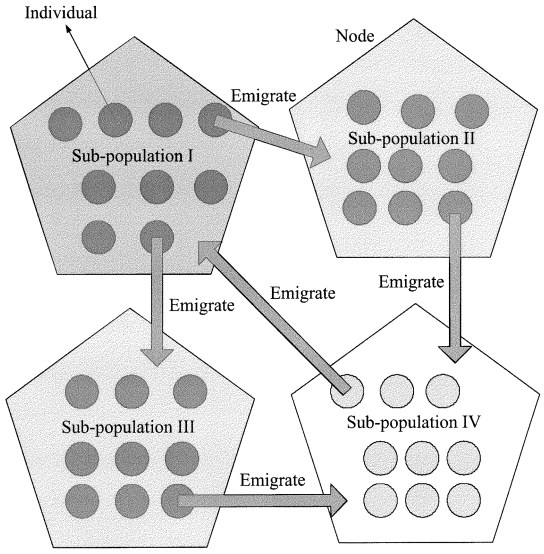

Fig. 1. Example of distributed evolutionary computing model.

evolutionary algorithms have been proposed to reduce the computational effort needed in solving complex optimization problems [4], [14]–[16]. Instead of evolving the entire population in a single processor, the parallel evolutionary algorithms applied the concept of multiple inter-communicating subpopulations in analogy with the natural evolution of spatially distributed populations [14]. Such inter-communication allows individuals to migrate among the subpopulations based upon some patterns to induce diversity of elite individuals periodically, in a way that simulates the species evolved in natural environment. These parallel evolutionary algorithms have been applied to solve many sophisticated problems in various fields, such as image processing [17], VLSI [18], network design [19], and drug scheduling [16].

As categorized by Rivera [6], there are four possible strategies to parallelize EAs, i.e., global parallelization, fine-grained parallelization, coarse-grained parallelization, and hybrid parallelization. In global parallelization, only the evaluation of individuals' fitness values is parallelized by assigning a fraction of the population to each processor to be evaluated. The genetic operators are often performed in the same manner as traditional EAs since these operators are not as time-consuming as the fitness evaluation. This strategy preserves the behavior of traditional EA and is particularly effective for problems with complicated fitness evaluations. The fine-grained parallelization is often implemented on massively parallel machines, which assigns one individual to each processor and the interactions between individuals are restricted into some neighborhoods. In coarse-grained parallelization, the entire population is partitioned into subpopulations. This strategy is more complex since it consists of multiple subpopulations and different subpopulations may exchange individuals occasionally (*migration*). As stated by Cantú-Paz [4], coarse-grained parallel EAs are more difficult to understand since the effects of migration are not fully understood. Besides, this strategy introduces fundamental changes in the EA operations and has a different behavior than traditional EAs [20]. In hybrid parallelization, several parallelization approaches are combined,

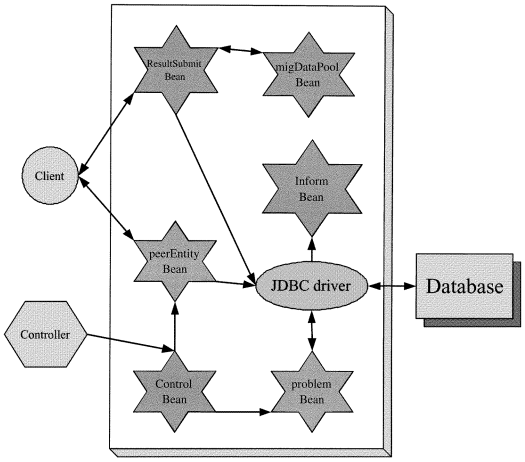

Fig. 2. Architecture overview of Paladin-DEC software.

and the complexity of these hybrid parallel EAs depends on the level of hybridization in the algorithm.

As shown in Fig. 1, the implementation of distributed evolutionary algorithms over the Internet can be extended from the model of coarse-grained parallel EAs with certain modifications in the features like migration scheme, task scheduling and fault tolerance, in order to adapt the model to a distributed computing environment. Unlike parallel evolutionary algorithms working in a well-defined infrastructure, a distributed evolutionary algorithm system must be able to bear with problems such as variant communication overhead, unpredictable node crash, network restrictions and other unpredictable events over the Internet. In such a distributed evolutionary computing system, genetic operations can be performed in each node, and the period of migration (migration interval) as well as the number of individuals migrate to other nodes (migration rate) can be fixed or adaptively determined along the evolution.

III. Distributed Evolutionary Computing Software

This section presents the architecture and implementation of a distributed evolutionary computing software named "Paladin-DEC," which has been developed based on the work of Tan *et al.* [16]. The software implements distributed evolutionary algorithms in a general framework of Java-based distributed system, which enhances the concurrent processing and performance of evolutionary computing by allowing inter-communications of subpopulations among multiple computers distributed over the Internet. The Paladin-DEC software aims to employ the resources of networked computers and inexpensive bandwidth to conquer complex optimization problems, which may be unsolvable or difficult-to-solve using a single computer.

A. Architecture Overview

The Paladin-DEC software is built upon the foundation of Java technology offered by Sun Microsystems and is equipped with application programming interfaces (APIs) and technologies from J2EE. The Java technology offers the unique advantages of platform independence and reusability, which are unmatched by most programming languages. The J2EE is a component-based technology provided by Sun Microsystems for design, development, assembly and deployment of enterprise applications. It offers various features such as multitier distributed application model, ability to reuse components, integrated extensible markup language (XML) based data interchange, unified security model and flexible transaction control. The J2EE has been widely used in large-scale e-commerce and enterprise applications, which is believed to be a suitable technology for developing the Paladin-DEC software based upon the architecture of multi-tier in J2EE. As shown in Fig. 2, the Paladin-DEC software consists of four main components, i.e., client, controller, enterprise Java beans (EJBs) and database. The EJB is a middle-tier component by which data are presented and business logics are performed. Different tiers are independent from each others and can be changed easily, e.g., changing the database or adding/removing certain business logics.

The clients are the main computational components in Paladin-DEC software, which perform computation according to the class loaded from an http server. As shown in the client interface in Fig. 3, users may upload their created job files through an ftp client integrated in the client program. As shown in the controller interface in Fig. 4, the controller functions as the control panel by which administrator can check the connections and jobs queued, to start, or stop a job. There are a total of 6 EJBs running in the application server, i.e., two entity beans, two stateful session beans, and two stateless session beans. These EJBs are the main components for performing different functions of the system which are described below.

The peerEntity bean is an entity bean for representing the peer's information, which is created when a client connects to the server and is removed when the client logoffs or timeouts. A peerEntity bean is uniquely identified by a valid email given by the user installing the client program in his/her computer. Once a peerEntity bean has been created, it waits for the controller to launch a task and to assign a task name. The problem bean is another entity bean in the system, which represents the job launched and the information combined. A problem bean is created when control bean launches a job from the job queue and is removed when the job is finished or terminated by the administrator. The control bean is a stateful session bean for the

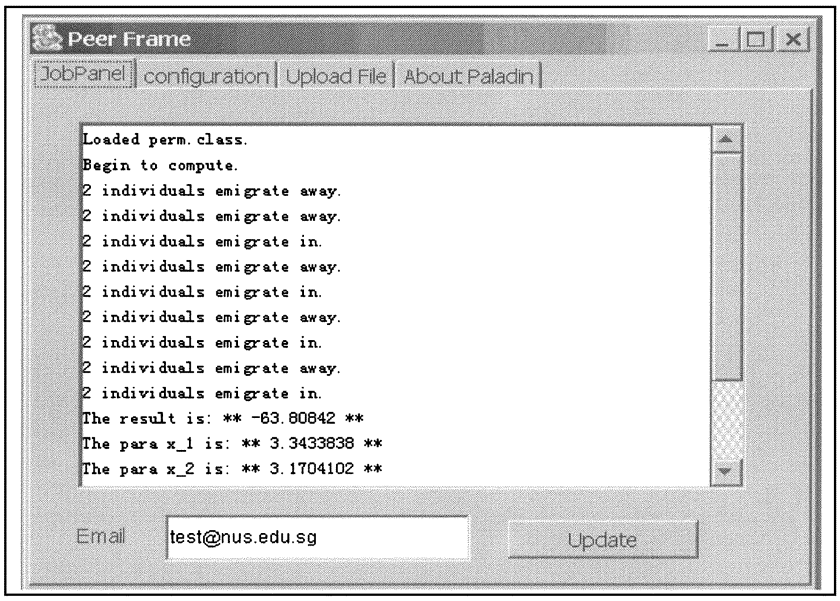

Fig. 3. Client user interface.

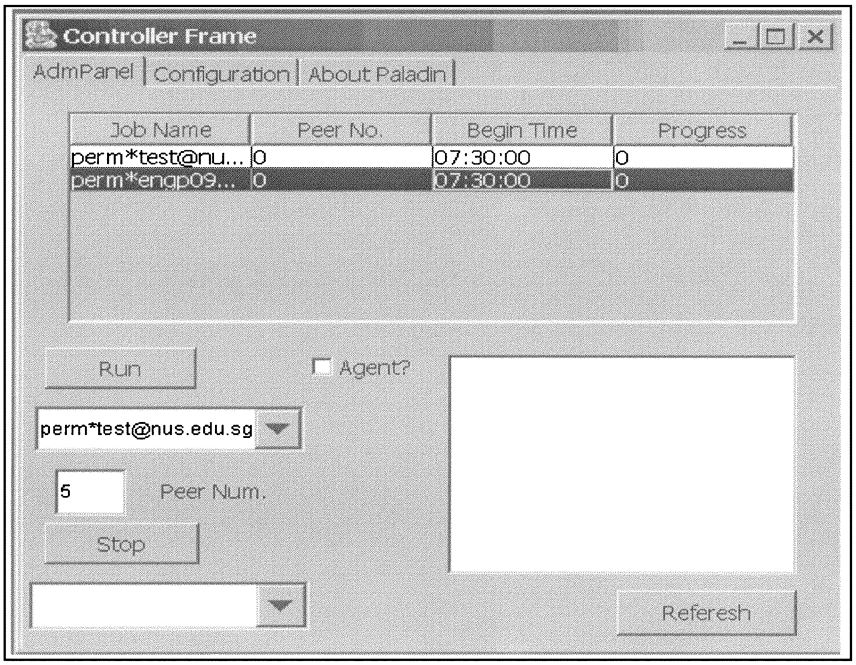

Fig. 4. Controller user interface.

controller program to manage the system. It is created when the administrator runs the controller program and is removed when the controller program is closed. Once the administrator decides to launch a job file, the control bean will be triggered to create a problem bean for the file. The peerEntity beans are then chosen from the bean pool and are assigned the task names and the corresponding http server address. It should be noted that a recommended peer number within the problem class file can be set by the user submitting a problem file and the administrator can change the number of the peerEntity beans before the task is actually launched.

The MigDataPool bean is the second stateful session bean for data migration when some tasks are running and migration operation is required. The migration is the core operation in Paladin-DEC software and different migration strategies could enhance or degrade the performance of the system. There are three options for the migration: migDataPool bean, Java message service (JMS) and peer agent. The migration through stateful session bean has good performance and reliability in the situation where not too many clients (e.g., below 40) are involved. A migDataPool bean is created or removed when a problem bean is formed or eliminated. It contains the intermediate individuals chosen to migrate to other subpopulations and hence is the core component of the distributed evolutionary system. The JMS is a Java API designed by Sun and several partner companies that allows different clients to exchange

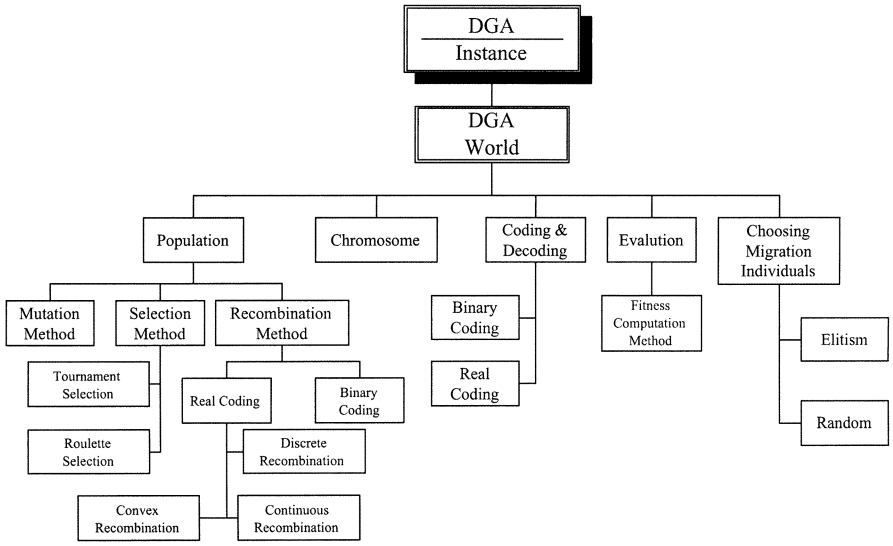

Fig. 5. Class hierarchy of DGA.

messages asynchronously and reliably. Due to the inherent parallelism of evolutionary algorithms, the JMS is a useful tool for implementing the migration function. It should be noted that both point-to-point (PTP) and publisher/subscriber (Pub/Sub) approaches can be used to perform the migration. Each job running in the system can be assigned a JMS queue by which clients running the same job can migrate individuals among each other.

The peer agent is extended from the work of Tan *et al.* [16], in which RMI-IIOP is used for migration and communication. The peer agent functions in the situation where massive communication is needed and many tasks are performed simultaneously. If the peer agent scheme is activated, peers chosen in a group can be assigned two roles, i.e., worker or agent. The worker performs computation and contacts agents for migration and results submission. The agent registers itself as an agent in the naming server and waits for the migration and results submission applications from workers in the same group. If any termination conditions are met, it will submit the best result to the server and dismiss the group. Although the scheme has many advantages, the agent chosen is not very robust since the server and backup agent scheme does not provide good robustness in the system. The crash or restart of individual agent computer can cause the problems of wasting resources or lost of results. The resultSubmit bean is a stateless session bean used to submit results and migration data from clients to server as well as to collect the migration data from server to clients. The inform bean is another stateless session bean used to collect the best result from all results submitted by the clients performing the same task as well as to inform the user who had submitted the job through email.

B. Algorithms

As mentioned in the Introduction, the Paladin-DEC software is developed for executing evolutionary algorithms in a distributed environment over the Internet for solving complex optimization problems. There are three different evolutionary algorithms incorporated in the software, i.e., distributed genetic algorithms (DGA), distributed genetic programming (DGP), and distributed evolutionary strategies (DES). Although these three algorithms have been developed based upon the basic principle of natural evolution, they are different in terms of implementation and functionality as described in the handbook of evolutionary computation [13]. Since the Paladin-DEC software provides a general framework for distributed computing, other types of evolutionary algorithms can be easily incorporated in the framework by expanding the class hierarchy if necessary.

Holland's pioneering book *Adaptation in Natural and Artificial Systems* [21] shows that evolutionary process can be applied to solve a wide variety of problems using a highly parallel technique that is now called the genetic algorithm. The genetic algorithm (GA) transforms a population of individual objects, each with an associated fitness value, into a new generation of the population using the Darwinian principle of reproduction and "survival-of-the-fittest." Each individual in the population represents a possible solution for the problem and is usually represented as a fixed-length binary-coded chromosome. The genetic algorithm attempts to find the near-optimal solutions to the problem by genetically breeding the population of individuals via the operations of selection, crossover, and mutation over a series of generations. The class hierarchy of DGA is shown in Fig. 5. For more details of genetic algorithms, readers may refer to the book of Goldberg [1].

Genetic programming (GP) has been developed according to the idea that desired program should evolve itself during the evolution process [22] and [23]. Instead of solving a problem or building an evolution program to solve the problem, one should search the space of possible computer programs for the best one. Genetic programming provides a way to run such a search by adopting a tree-based chromosome structure for genetic representation. The DGP in Paladin-DEC software was developed based on the GNU public domain software of groovy Java genetic programming (JGProg) package available at http://jgprog.sourceforge.net. The class hierarchy of DGP is shown in Fig. 6. Readers may refer to the books of Koza

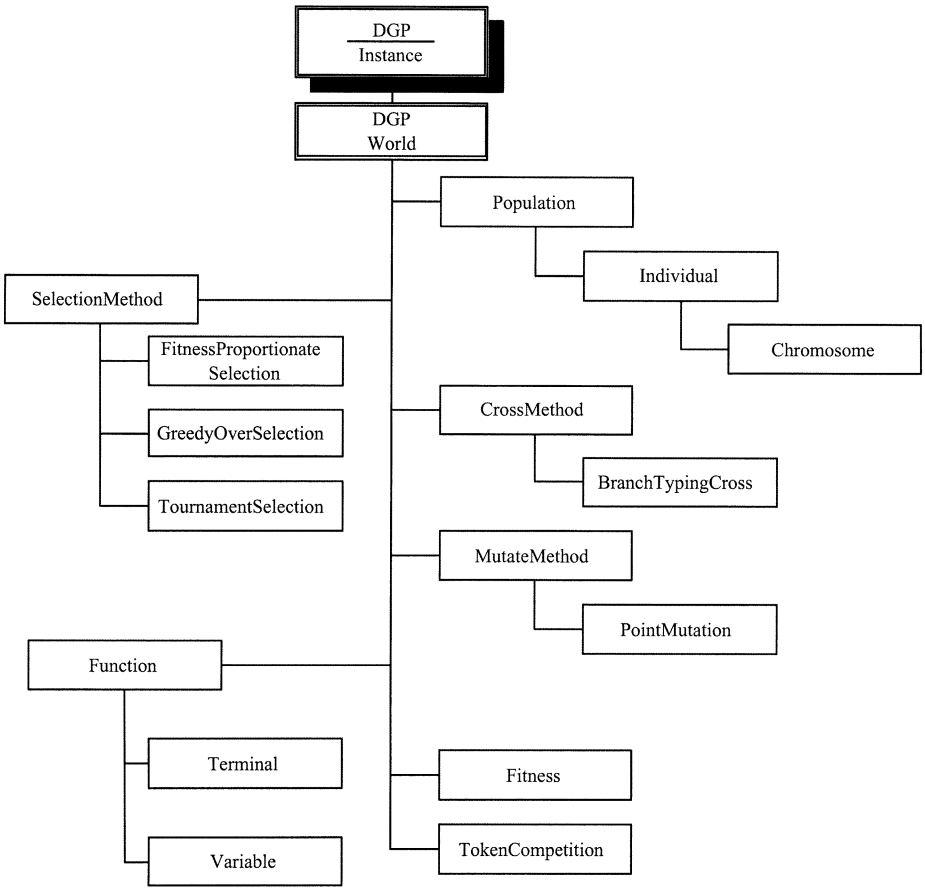

Fig. 6. Class hierarchy of DGP.

[22], [23] for more details of genetic programming and its applications.

The evolutionary strategies (ES) algorithm has been developed for solving real-parameter optimization problems based upon a single genetic mutation operator. In ES, a chromosome represents individual as a pair of float-valued vectors, i.e., $\vec{v} = (\vec{x}, \vec{\sigma})$, where the first vector $\vec{x}$ represents a point in the search space and the second vector $\vec{\sigma}$ is a vector of standard deviations. The mutations are realized by replacing $\vec{x}$ by $\vec{x}^{i+1} = \vec{x}^{i} + N(0, \vec{\sigma})$, where $N(0, \vec{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviation $\vec{\sigma}$. The offspring is accepted as a new member of the population if it has better fitness and all constraints are satisfied. The main idea behind these strategies is to allow the control parameters to self-adapt rather than changing their values in a deterministic way. The class hierarchy of DES is shown in Fig. 7. Readers may refer to the works of Back *et al.* [13] and Schwefel [24] for more details of evolutionary strategies.

C. Workflow

The working process of a client is shown in Fig. 8 which begins when a client is started and logons to the server. A peerEntity bean, uniquely identified by a valid email address, is then created and pooled. The client will check the status at regular intervals to see whether it has been assigned a job. Once the client detects that its corresponding bean is updated due to the assignment of a job, it will read the information from the bean, as well as extract class name, path, and http server address before loading the class remotely from the server. If the class loaded is consistent with the specification of the system, the computation procedure will be initiated and the client will check whether the instance needs any migration at each generation. If the conditions for migration are fulfilled, the client will initiate a session with the resultSubmit bean in the server and choose some individuals according to the migration rate. It will then send the data to the migDataPool bean through the resultSubmit bean and obtain the same number of individuals from the server for migration. If a running job is cancelled by the controller, those clients involved in the job will stop the computation and resent to the ready status. If any client meets the termination conditions, it will initiate a session with the resultSubmit bean and submit the results before restoring to the ready status.

The working process of the server is shown in Fig. 9. If the controller launches a queuing job, the corresponding control bean will create a problem bean uniquely identified by the class name combined with the email of the submitter. A migDataPool bean is then created by the problem bean to deal with the migration work in this task. After creating the problem bean, the control bean will select peers from the pool and assign to them the information about the class and http server from which the client can load the problem class. After a client finishes the job, it will submit the results to the server and set itself to the ready

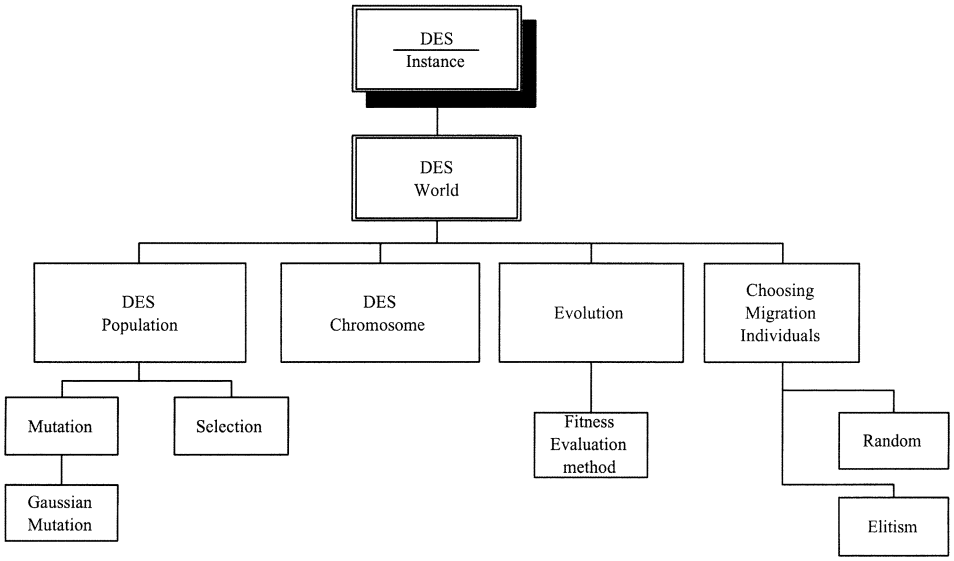

Fig. 7. Class hierarchy of DES.

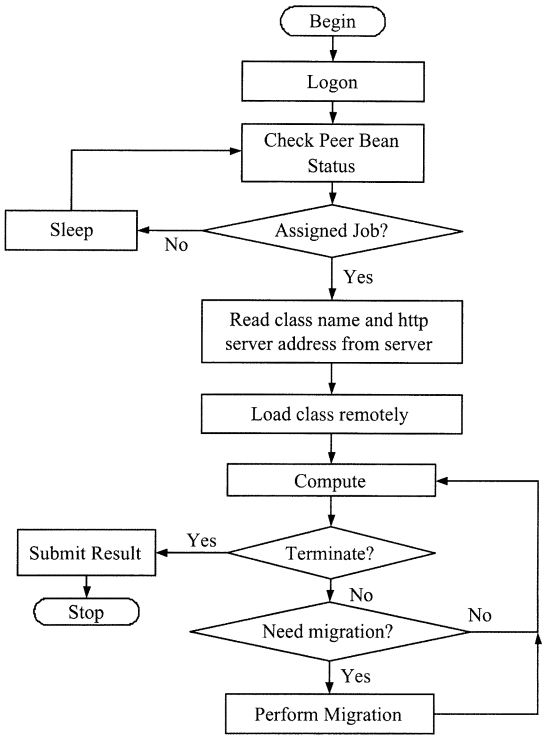

Fig. 8. Working process of a client.

status. If all the clients finish their jobs and submit the results or any terminating conditions have been met, the problem bean will initiate the inform bean that will choose the best result from all the results obtained and send it to the user submitting the job via email. Subsequently, the problem bean will remove the attached migDataPool bean and inform the control bean. The control bean then removes the problem bean and updates those peers involved to ensure that all the peers are restored to the ready status. At this stage, all the peerEntity beans will be ready and pooled for the next job.

The Paladin-DEC software requires the setup of Java 2 platform standard edition (J2SE) 1.4.0 or higher version and J2EE 1.3.0 or higher version for building up the Java environment. The http server and ftp server are also necessary in order to receive the problem files uploaded from the users. Besides, a database server is required for which a script is provided to create the necessary database and tables. A template designed for writing the problem file that specifies the function evaluation and optimization variables corresponding to the user's particular problem is also included in the software. The problem class in the file implements two methods as its prerequisite, i.e., *setPara*() that contains a few subsections to control the whole process of the computation and *evaluateFuc*() that is an intensively used method to evaluate the fitness of individuals in the evolution process.

D. Work Balancing

Since the processing power and specification for various computers in a network might be different, the feature of work balancing that ensures the peers are processed in a similar pace is needed in a distributed evolutionary system. This is important because peers that finish the computation relatively faster than others may quit from the system early, which could significantly affect the structure of inter-communications among multiple subpopulations in a distributed evolutionary algorithm. Intuitively, work balancing for a distributed evolutionary system could be difficult due to the fact that the working environment in a network is often complex and uncertain. The Paladin-DEC software incorporates a simple work balancing strategy for which subpopulation size in different peers with different computational power is adjusted adaptively based upon the general principle of assigning larger/smaller number of subpopulations to faster/slower machines. In this approach, the system computes the process status of each peer at every 10% of the generation size and adjusts the subpopulation size as necessary. The schemes of reshuffle and best-worst have been used in the Paladin-DEC software to reduce the computation gaps between peers in the same group.

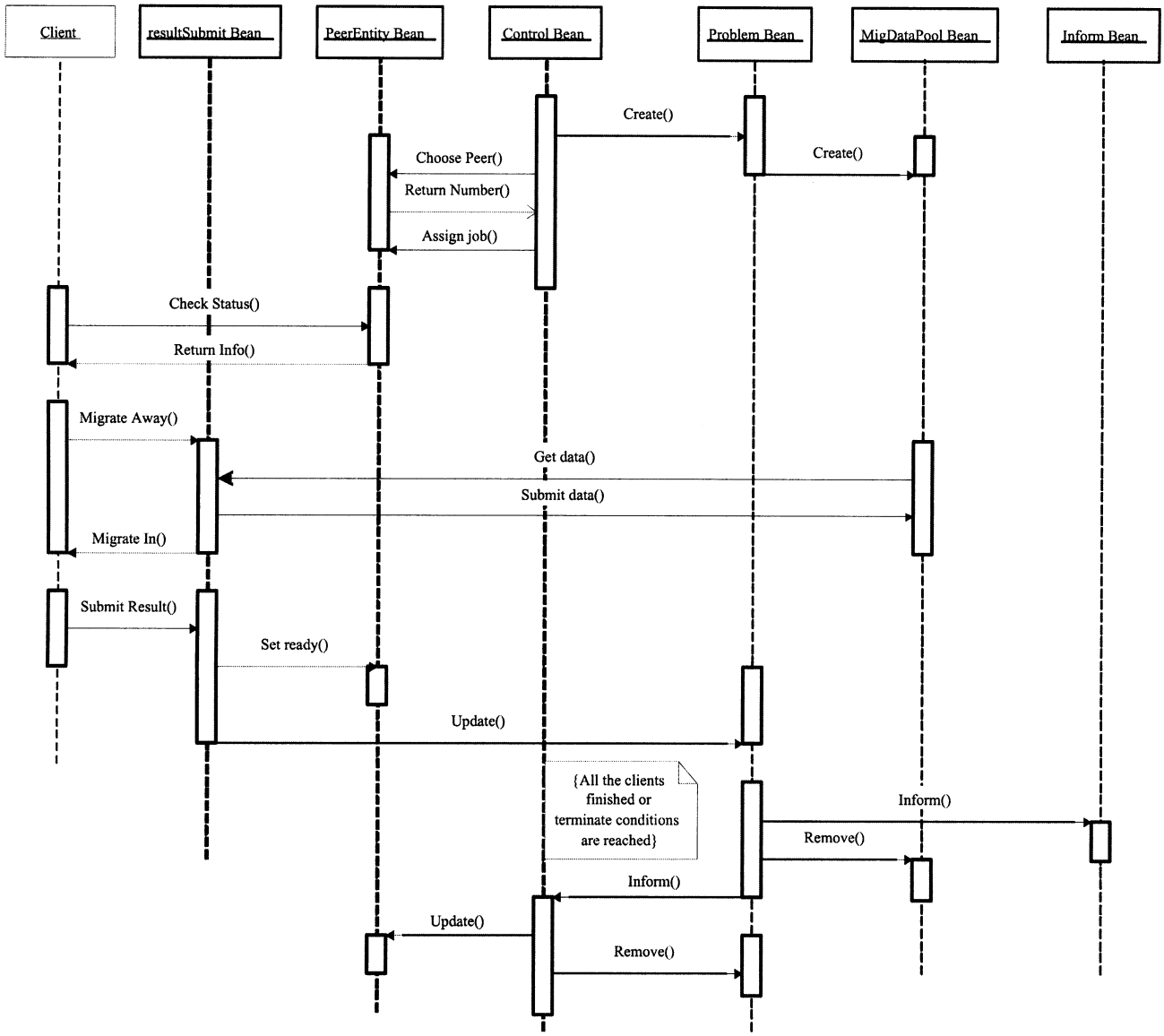

Fig. 9. Working process of server.

In the reshuffle scheme, every peer will be assigned the same amount of individuals in the subpopulations initially. At any checkpoint, the system first normalizes the process according to $Pro(i)/\sum_{i=1}^{n} Pro(i)$ and removes any peers that cannot reach a set threshold. It then reshuffles the subpopulation size of each peer according to $Pz \cdot Pro(i)/\sum_{i=1}^{n} Pro(i)$, where Pz is the total number of individuals in the whole groups; $Pro(i)$ denotes the process status of each peer in the check point; n is the number of peers in a group and n' is the remaining number of peers in the group. The peers that have been removed or having reduced population size will migrant its individuals to the migDataPool bean, which then migrates an appropriate amount of individuals to the peers that should be assigned more individuals according to the above computation results. The best-worst scheme follows the same procedure as the reshuffle scheme before the reshuffle process. After deleting those peers below the threshold, the best-worst scheme chooses the fastest peer and the slowest peer in the remaining peers. If the fastest peer is more than 10% faster than the slowest peer, 10% of the individuals from the slowest peer will be moved to the fastest peer directly. Empirically this scheme is more efficient and capable of saving significant communication cost for a small group of peers.

E. Robustness, Portability, and Security

Robustness in software development is often an important issue, particularly for distributed systems with complex or uncertain working environment among multiple computers over the Internet. A key factor in Java programming is exception handling where any action that can trigger an exception must be caught *a-priori* and the corresponding codes that can safely handle the exception to ensure the smooth working of the system should be implemented. The data integrity in a system will be lost if multiple programs are allowed to update the same information simultaneously or when a system failed while processing

a business transaction and left the affected data to be only partially updated. By preventing both of these scenarios, software transactions ensure data integrity by controlling the concurrent access of data via multiple programs. In the event of a system failure, transactions make sure that the data will be in a consistent state after the recovery. In the Paladin-DEC system, a single peer crash causes little influence to the overall results since the elite individuals migrate among all the participated peers that fortify the integral robustness of the system. A pre-examined function is also incorporated in the Paladin-DEC software to validate all problem files before allowing these files to be uploaded to the server in order to avoid any fatal bugs that could crash programs in the peers.

The J2EE technology is a set of standards that brings the advantage of write once, run anywhere (WORA) to the system. By using the software, the program can run on any platform with Java support and the user can choose any application server available that follows the standards without any extra configuration. The Java remote method invocation (Java RMI) technology runs over the Internet inter-orb protocol (RMI-IIOP) delivers the common object request broker architecture (CORBA) distributed computing capabilities to the Java 2 platform. Similar to CORBA, the RMI over IIOP is based on open standards defined with the participation of hundreds of users in the object management group (OMG). The OMG has worked diligently to provide a greater level of portability by tightening up the CORBA specification.

Security is one of the key attributes that make Java technology a superior programming environment. There are features in the language, virtual machine and core class libraries that facilitate the creation of secure applications. The Java virtual machine (JVM) relies on the software technology to present a secure sandbox where the Java program runs. This sandbox verifies all classes coming into the sandbox and performs numerous runtime checks to ensure the Java program does not perform invalid operations. Besides, it also presents as a fence around the runtime environment to control all access outside the sandbox. The J2EE application programming model has insulated developers from the mechanism-specific implementation details of application security in a way that enhances the portability of applications and allows them to be deployed in diverse security environments.

IV. Case Studies of Paladin-Dec Software

To illustrate the advantage of Paladin-DEC software, the problems of drug scheduling in cancer chemotherapy and searching probe sets of yeast genome have been studied. These problems are difficult to solve and often require an extremely long computation time for traditional evolutionary algorithms running in a single computer. Using the Paladin-DEC software, however, a large population size could be used to extend the search power of an evolutionary algorithm and the computational workload can be shared among multiple computers over the Internet. The logon server for the case studies runs in a Linux server and the dispatcher server runs in a PC with Windows 2000 operating system. The peers are computers connected within the university network, which have been installed with different operating systems, including Windows ME, Windows 98 SE, and Windows 2000.

A. DGA for Drug Scheduling in Cancer Chemotherapy

1) Drug Scheduling Problem: An important problem in cancer chemotherapy is to determine the control policy such that after a fixed period of therapy, the tumor burden is minimized. This is essential when surgery, or laser treatment, is scheduled at some future date. The effects of cancer chemotherapy, as outlined by Martin [25], are given by the following differential equations

$$\frac{dx_1}{dt} = -\lambda x_1 + k(x_2 - \beta)H(x_2 - \beta) \tag{1}$$

$$\frac{dx_2}{dt} = u - \gamma x_2 \tag{2}$$

$$\frac{dx_3}{dt} = x_2 \tag{3}$$

with the initial state $\boldsymbol{x}^T(0) = [\ln(100) \quad 0 \quad 0]$ and

$$H(x_2 - \beta) = \begin{cases} 1, & \text{if } x_2 \geq \beta \\ 0, & \text{if } x_2 \leq \beta \end{cases} \tag{4}$$

where x_1 is a transformed variable that is inversely related to the mass of the tumor. The tumor mass is given by $N = 10^{12} \cdot e^{(x_1)}$ cells and the initial tumor cell population is set as 10^{10} cells [25]. The variable x_2 is the drug concentration in the body in drug units $[D]$ and x_3 is the cumulative effect of the drug.

Equation (1) describes the net change in the tumor cell population per unit time. The first term on the right-hand side of the equation describes the increase in cells due to cell proliferation, and the second term describes the decrease in cells due to the drug. The parameter λ is a positive constant related to the growth function; k is the proportion of tumor cells killed per unit time per unit drug concentration which is assumed to be a positive constant. Equation (2) describes the net increase in the drug concentration at the cancer site. The variable u is the rate of delivery of the drug, and the half-life of the drug is $\ln(2)/\gamma$. Equation (3) relates the cumulative toxicity of the drug to the drug concentration, e.g., the cumulative effect is the integral of the drug concentration over the period of exposure. The implication of the function described in (4) is that there is a threshold drug concentration level, β, below which no tumor cells are killed. The values of the parameters used are $\lambda = 9.9 \times 10^{-4}$ (per day), $k = 8.4 \times 10^{-3}$ (per day per D), $\beta = 10$ (D), and $\gamma = 0.27$ (per day). The performance index PI to be maximized is [25]

$$PI = x_1(t_f) \tag{5}$$

where the final time $t_f = 84$ days. The control optimization is performed subject to the following constraints

$$u \geq 0 \tag{6}$$

$$x_2 \leq 50 \tag{7}$$

$$x_3 \leq 2.1 \times 10^3. \tag{8}$$

The constraints on the drug concentration, x_2, and the cumulative drug effect, x_3, in (7) and (8) are to ensure that the patient can tolerate the toxic side effects of the drug. In order to reduce

TABLE I
AVERAGE RESULTS FOR DIFFERENT NUMBER OF PEERS (DIFFERENT SUBPOPULATION SIZE)

Number of peers	Subpop. size	Case I (Generation = 1000)		Case II (Generation = 500)	
		Time (s)	*PI*	Time (s)	*PI*
1	2000	1033	16.8826	392	17.7197
2	1000	565	16.9223	314	17.6874
4	500	421	16.8664	159	17.6819
5	400	308	16.8890	130	17.7577

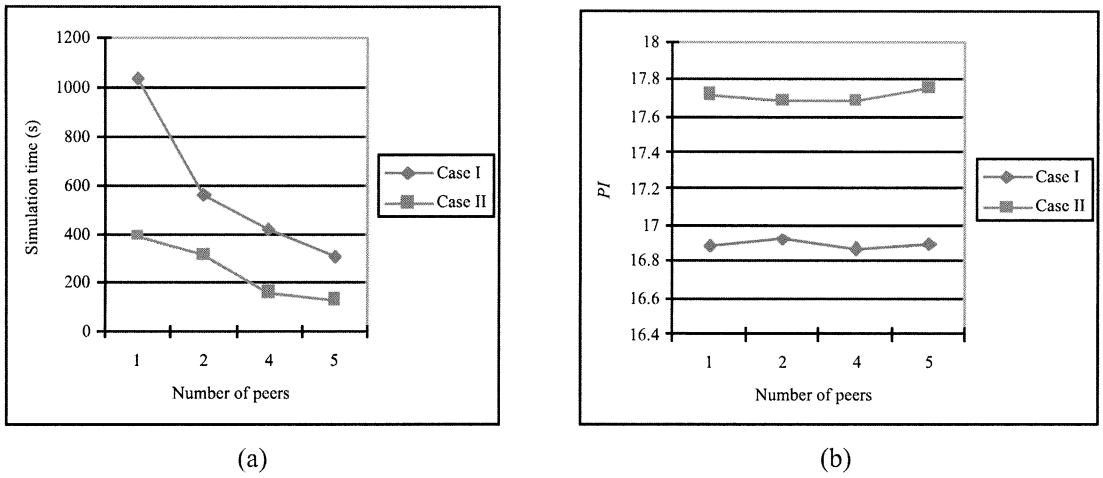

Fig. 10. Average simulation time and performance index versus number of peers.

the likelihood of the emergence of drug-resistant cells, the tumor size is forced to reduce by at least 50% every three weeks, so that $x_1(21) \geq \ln(200)$, $x_1(42) \geq \ln(400)$, and $x_1(63) \geq \ln(800)$.

2) Drug Scheduling via DGA: An evolutionary algorithm has been applied to solve the problem of drug scheduling in cancer chemotherapy using a simplified distributed computing architecture [16]. In this paper, we are interested in studying how the DGA running in a distributed computing environment affects the overall scheduling performance and computational speed as compared to traditional GA running on a single computer. Without loss of generality, the DGA in Paladin-DEC has been applied to solve the drug scheduling problem for different number of peers ranging from 1–6 with similar genetic parameter settings: tournament selection size = 2; crossover rate = 0.7; mutation rate = 0.1; migration rate = 0.01; migration interval = 10; number of preserved elite individuals = 50. It should be noted that these parameter values have been chosen based on the common settings of genetic algorithms, which are neither fine-tuned, nor adjusted to their optimal values for the different setting of peer numbers.

The fitness function [16] is denoted by F_{point} and F for the problem with and without point constraints, respectively

$$F_{\text{point}} = PI - p_{\text{state}} - p_{\text{point}} \tag{9a}$$

$$F = PI - p_{\text{state}} \tag{9b}$$

where, $p_{\text{state}} = \sum_{i=2}^{3} w(x_i) \cdot (x_i - \overline{x}_i)$; $p_{\text{point}} = w(x_1) \sum_{j=1}^{3} (x_1(j) - \overline{x}_1(j))$; $\overline{x}_1(1) = \ln(200)$; $\overline{x}_1(2) = \ln(400)$; $\overline{x}_1(3) = \ln(800)$; $w(x_1) = 2$; $w(x_2) = 1$; $w(x_3) = 0.3$; $\overline{x}_2 = 50$; $\overline{x}_3 = 2100$; and PI is the performance index as given in (5). The weight w has been chosen such that the value of $W(x_i) \cdot \overline{x}_i$ is about equal, for $i = 1$, 2, and 3, in order to distribute the penalty effect resulting from each of the constraints.

The cancer chemotherapy model was simulated using the numerical differentiation method of Runga-kutta [26] with a small time interval of 0.01 day for good accuracy. We performed ten independent runs with random initial population to minimize any bias in the simulations. The pair-wise variable representation scheme [16] which encodes the information of dosage level and start-day as variable pairs is adopted here (eight pairs for case I and five pairs for case II). The average simulation results are shown in Table I and Fig. 10, where case I is for the problem with all constraints satisfied and case II with the 3 state constraints omitted. By keeping the overall population size constant at 2000 and with different subpopulation size according to the number of peers ranging from 1–5, the DGA can dramatically reduce the computation time without sacrificing scheduling performance as the number of peers is increased. For example, in case I, the average computation time for five peers (each peer with 400 individuals) is 308 s, which is more than three times shorter than the 1033 s needed for one peer (with 2000 individuals), although they both achieved a similar performance index of 16.88.

As shown in Table II and Fig. 11, for the simulations with different number of peers ranging from 1–6 and each peer with

TABLE II
AVERAGE RESULTS FOR DIFFERENT NUMBER OF PEERS (CONSTANT SUBPOPULATION SIZE OF 1000)

Number of peers	Case I (Generation = 1000)		Case II (Generation = 500)	
	Time (s)	*PI*	Time (s)	*PI*
1	933	16.8826	392	17.7197
2	1029	16.9333	422	17.7992
3	1037	17.1405	358	17.8163
4	1095	17.1422	405	17.8264
5	1150	17.1965	405	17.8625
6	1061	17.2694	428	17.8710

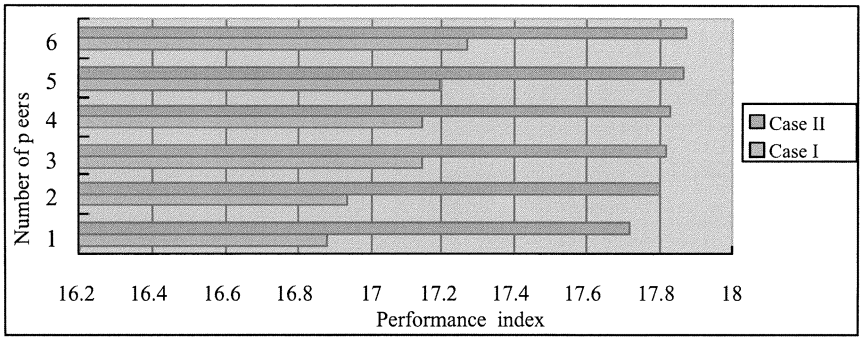

Fig. 11. Comparisons of average performance index for different number of peers.

a constant subpopulation size of 1000, the average scheduling performance is improved as the number of peers is increased although the computation time needed remains the same. For example, in case I, the average performance index is 17.26 for six peers (each peer with 1000 individuals), which is much better than the average performance index of 16.88 for one peer (with 1000 individuals) in this maximization problem, although they both required a similar computation time (e.g., 1061 s for six peers and 933 s for one peer). The slight extra time needed for six peers (as compared to one peer) is mainly due to the intercommunications overhead for migration, workload balancing and results collection among the multiple computers over the Internet. One observation from case II in Table II is that the computation time needed for two peers appears to be larger than that of three–five peers. This result is because the communication status and speed of the Internet often vary from time to time, which may cause minor fluctuation in terms of computation time for a distributed system occasionally.

B. DES for Searching Probe Sets of Yeast Genome

1) Background: The DNA microarray, also known as DNA CHIP, is a revolutionary technology that involves immobilization of a large numbers of different DNA molecules within a small confined space [27] and [28]. Over the years, several technologies have been developed to attach DNA molecules to solid platform. Oligonucleotides (short single stranded DNA molecules) can be synthesized in situ using photolithographic techniques or phosphoramidite chemistry by ink jet printing technology [29] and [30]. The precision of photolithographic technology allows the synthesis of high resolution and extremely high density DNA microarrays. Such arrays are currently marked by Affymetrix. Alternatively, DNA molecules, typically in the form of double stranded PCR (polymerase chain reaction) products or oligonucleotides, can be attached to glass slides or nylon membranes [31]. The latter method is a more practical and cost-effective avenue of making DNA microarrays by most standard laboratories. In addition, it offers the flexibility of printing DNA of choice onto solid platform.

The stability and association between complementary DNA molecules critically depend on the melting temperature T_m. The T_m is operationally defined as the temperature at which 50% of a single stranded DNA annealed with its complement to form a perfect duplex. The T_m is governed by several factors: base composition, DNA concentration, salt concentration, and the presence of destabilizing chemical reagents. As an GC base pair is held together by three hydrogen bonds while an AT base pair has only two hydrogen bonds, the GC rich sequence has a higher T_m as compared with the AT rich sequence. A higher concentration of DNA favors duplex formation and consequently the T_m is higher. As cations stabilize DNA duplexes, higher salt concentration raises the T_m. Chemicals such as formamide or DMSO destabilize DNA duplexes and therefore have a negative effect on T_m. In a typical microarray experiment, thousands of DNA spots on the microarray interact with a very complex mixture of labeled DNA under a single condition. Therefore an optimal hybridization condition is necessary to obtain the best result, and one way to attain the optimal hybridization is to control the T_m of the immobilized DNA on the microarray.

TABLE III
AVERAGE RESULTS FOR SEARCHING THE PROBE SET OF AN ORF (6310)

Number of peers	Avg. computation time (seconds)	Melting temperature (°C)	Avg. success rate over 5 independent runs
1	220	74.3690	20%
2	132	74.3965	40%
3	45	74.9451	100%

The yeast Saccharomyces cerevisiae is the first eukaryote genome that has been sequenced which has approximately 6000 genes. The gene structure of this yeast is relatively simple as compared to higher eukaryotes. For example, there are very few genes contain introns and most of the open reading frames (ORF), which are protein coding sequences, are preceded by promoters. Since detailed sequence information is known for all predicted gene in this organizm, the Paladin-DEC software is applied to find the unique DNA sequences with optimized melting temperature that can be printed onto the DNA microarrays. The yeast is simpler both in its behavior and its genome structure as compared to complex Vertebrates [32]. Yeast genomics remains an interesting area of research, as most biologists are concerned with the information and clues extracted from the yeast DNA array, and the eventual goal is to search for the probe set of the human genome that is currently not available. One of the main limitations or obstacles in using the microarray is that ORF's are extremely variable in length and T_m (T_m is the melting point of the particular ORF), making comparison between any two genes on the array virtually impossible. The problem is thus to search for probes within each ORFs so that the probes are unique, approximately the same length and melting temperature.

2) Fitness Function: There are three criteria for a qualified sequence: uniqueness of the sequence, the sequence should have a melting temperature within a special range, and the sequence should not have any complementary part that could cause folding back of the sequence. A qualified probe/sequence is thus one that satisfies all these three criteria. The pseudo-code for calculating the fitness function is given as follows (the fitness scale is chosen empirically).

a) Uniqueness criterion: As discussed, the qualified sequence/probe should not appear in other ORFs. There are two main characteristics of the uniqueness criterion. First, the computational cost of the uniqueness test is substantially high. A sequence/probe is first determined by randomly choosing two base-pairs from the ORF, i.e., the start point and the end point. To determine whether one sequence appeared in this long database is a computationally expensive task. Second, the feasible region of the sequences that satisfy the uniqueness condition is highly nonlinear. Some genes share high degree of sequence conservation because they are evolved from a common ancestor [33]. Even for those nonrelated sequences, they are still some similar subsequences having the same functions [33]. These sequences are distributed all over in the ORF, making the feasible region discrete and nonlinear.

b) Melting temperature criterion: The melting temperature (T_m) of an oligonucleotide is referred to the temperature at which the oligonucleotide is annealed to 50% of its exact complement. This temperature is directly related to a wide variety of applications including PCR, hybridization and anti-gene targeting. For subsequent processing using the microarray, the probes or subsequences should have a T_m in the specific range. A number of methods exist for the calculation of T_m and one of the more accurate equations for T_m is the nearest neighbor method [34],

$$T_m = \frac{H}{S + R \cdot \ln(\frac{C}{4})} + 16.6 \cdot \log \frac{[K+]}{1 + 0.7[K+]} - 273.15 \quad (10)$$

where H and S is enthalpy and entropy for helix formation, respectively. They represent the sum of the values of the nearest pair bases. For example, $H(GATC) = H(GA) + H(AT) + H(TC)$. The table of H and S values can be found in [34]. R is the molar constant, C is the concentration of the probe and $[K+]$ is the concentration of the $K+$. In searching for the qualified subsequence, R is set as 1.987 cal/(°C mol), $K+$ is equal to 50 mM and C is equal to 250 pM. A suitable T_m is chosen in the range of 65°C to 80°C.

c) Non folding-back criterion: A qualified subsequence must not have long complementary pair parts, which may cause self-folding and disturb the micro-array test. This occurs if a section of the subsequence/probe contains the complement of another section within the same probe, e.g., A.C.C.G.G and C.C.G.G.T. The longer the complementary pair appears, the more likely the folding-back occurs. The parameter (specifying the length of complementary pair) of the nonfolding test is set as seven.

3) Simulation Results: The 6310 ORFs used for the simulation are presented in plain text file, which are obtained from the Web site (http://genome-www.stanford.edu/Saccharomyces/lists_tables.html). An orf (no. 6310 > ORFN:YPR204W YPR204W, Chr XVI from 944 598-947 696), which has a difficult-to-search probe set, is cited as the candidate for comparison in the simulation. Without loss of generality, the DES in the Paladin-DEC software has been applied to find the probe set for this orf using different number of peers ranging from one–three with the following parameter settings: generation size = 400; subpopulation size = 150 (each peer contains 150 individuals); mutation rate = 0.1; migration rate = 0.02; migration interval = 40; lower bound of melting temperature = 65; and higher bound of melting temperature = 80.

The average results for five independent simulation runs with random initial population are listed in Table III. As can be seen, while the melting temperature constraint has been satisfied for

different number of peers ranging from one-three (e.g., in the range of 65°C to 80°C), the average computation time needed is 45 s for three peers, which is much shorter than the 220 s required for the case of one peer. Besides, the average success rate of finding a qualified sequence over the five simulation runs is 100% for three peers, which is much higher than the 20% success rate obtained for the case of one peer (e.g., only 1 qualified sequence is found out of the five runs/trials). It should be noted that the DES has managed to find the qualified sequences for all the 6310 ORF's and the results obtained are consistent with the ones shown in Table III. This illustrates the advantages of applying distributed evolutionary computing approach for searching the probe sets of yeast genome, i.e., the DES can dramatically decrease the computation time and increase the possibility of finding a qualified sequence, which could be a potential tool for finding more complicated genomes such as the human genome.

V. Conclusions

This paper has presented a Java-based distributed evolutionary computing software (Paladin-DEC), which enhances the concurrent processing and performance of evolutionary algorithms by allowing inter-communications of subpopulations among various computers over the Internet. Such a distributed system has enabled individuals to migrate among multiple subpopulations according to some patterns to induce diversity of elite individuals periodically, in a way that simulates the species evolve in natural environment. The Paladin-DEC software is capable of keeping data integrity throughout the computation, and has been incorporated with the features of robustness, security, fault tolerance and work balancing. The effectiveness and advantages of the Paladin-DEC have been illustrated upon two case studies of drug scheduling in cancer chemotherapy and searching probe sets of yeast genome. On-going work includes the development of peer-to-peer (p2p) computing using JXTA (Juxtapose) technology to improve the performance of the distributed system. The use of advanced application server such as BEA Weblogic could also enhance the performance and scalability of the system, and features of the server such as cluster and integrated Java message service could be explored to further enhance the system.

References

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley, 1989.

[2] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolutionary Programs*, 2nd ed. Berlin, Germany: Springer-Verlag, 1994.

[3] K. C. Tan, T. H. Lee, D. Khoo, and E. F. Khor, "A multi-objective evolutionary algorithm toolbox for computer-aided multi-objective optimization," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 537–556, 2001.

[4] E. Cantú-Paz, "A survey of parallel Genetic Algorithms," in *Calculateurs Paralleles, Reseaux et Systems Repartis*. Paris, France: Heimes, 1998, vol. 10, pp. 141–171.

[5] D. E. Goldberg, "Sizing populations for serial and parallel genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, San Mateo, CA, 1989, pp. 70–79.

[6] W. Rivera, "Scalable parallel genetic algorithms," *Artif. Intell. Rev.*, vol. 16, pp. 153–168, 2001.

[7] D. Levine. (1995) Users Guide to the PGAPack Parallel Genetic Algorithm Library ANL-95-18. [Online]. Available: http://www.mcs.anl.gov/pgapack.html

[8] D. Andre and J. R. Koza, "Parallel genetic programming on a network of transputers," in *Proc. Workshop Genetic Programming: From Theory RealWorld Applications*, vol. 95–2, 1995, pp. 111–120.

[9] M. Tomassini and F. Fernandez, "An MPI-based tool for distributed genetic programming," in *Proc. IEEE Int. Conf. Cluster Computing*, 2000, pp. 209–216.

[10] I. Tanev, T. Uozumi, and K. Ono, "Parallel genetic programming: Component object-based distributed collaborative approach," in *Proc. 15th Int. Conf. Information Networking*, 2001, pp. 129–136.

[11] F. S. Chong, "A Java Based Distributed Genetic Programming on the Internet," M.S. thesis, School of Computer Science, Univ. Birmingham, Birmingham, AL, 1997.

[12] B. Paechter and T. Back, "A distributed resources evolutionary algorithm machine (DREAM)," in *Proc. IEEE Congress Evolutionary Computation*, vol. 2, 2000, pp. 951–958.

[13] *Handbook on Evolutionary Computation*, T. Back, D. B. Fogel, and Z. Michalewicz, Eds., New York: Oxford Univ. Press, Bristol, U.K., 2000.

[14] V. Cristea and G. Godza, "Genetic algorithms and intrinsic parallel characteristics," in *Proc. IEEE Congr. Evolutionary Computation*, vol. 1, 2000, pp. 431–436.

[15] J. Nang and K. Matsuo, "A survey on the parallel genetic algorithms," *J. Soc. Instrum. Contr. Eng.*, vol. 33, no. 6, pp. 186–191, 1994.

[16] K. C. Tan, E. F. Khor, J. Cai, C. M. Heng, and T. H. Lee, "Automating the drug scheduling of cancer chemotherapy via evolutionary computation," *Artif. Intell. Med.*, vol. 25, no. 2, pp. 169–185, 2002.

[17] Y. W. Chen, Z. Nakao, and F. Xue, "A parallel genetic algorithm based on the island model for image restoration," in *Proc. 13th Int. Conf. Pattern Recognition*, vol. 3, 1996, pp. 694–698.

[18] N. Yoshida and T. Yasuoka, "Multi-GAP: Parallel and distributed genetic algorithms in VLSI," in *Proc. IEEE Int. Conf. Systems, Man, Cybernetics*, vol. 5, 1999, pp. 571–576.

[19] A. Sleem, M. Ahmed, A. Kumar, and K. Kamel, "Comparative study of parallel vs distributed genetic algorithm implementation of ATM network environment," in *Proc. 5th IEEE Symp. Computers Communications*, 2000, pp. 152–157.

[20] R. Subbu and A. C. Sanderson, "Modeling and convergence analysis of distributed co-evolutionary algorithms," in *Proc. IEEE Congress Evolutionary Computation*, vol. 2, 2000, pp. 1276–1283.

[21] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[22] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press, 1994.

[23] J. R. Koza, F. H. Bonnett III, D. Andrew, and M. A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann, 1999.

[24] H. P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, 1995.

[25] R. B. Martin, "Optimal control drug scheduling of cancer chemotherapy," *Automatica*, vol. 28, pp. 1113–1123, 1992.

[26] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Receipes in C: The Art of Scientific Computing*, 2nd ed. New York: Cambridge Univ. Press, 1992.

[27] R. J. Lipshutz, S. P. Fodor, T. R. Gingeras, and D. J. Lockhart, "High density synthetic oligonucleotide arrays," *Nat. Gene.*, vol. 21, no. Supplement 1, pp. 20–24, 2000.

[28] D. J. Lockhart and E. A. Winzeler, "Genomics, gene expression and DNA arrays," *Nature*, vol. 405, no. 6788, pp. 827–836, 2000.

[29] T. R. Hughes *et al.*, "Expression profiling using microarrays fabricated by an ink-jet oligonucleotide synthesizer," *Nature Biotech.*, vol. 19, no. 4, pp. 342–347, 2001.

[30] A. C. Pease, D. Solas, E. J. Sullivan, M. T. Cronin, C. P. Holmes, and S. P. Fodor, "Light generated oligonucleotide arrays for rapid DNA sequence analysis," in *Proc. Nat. Academy Sciences*, vol. 91, 1994, pp. 5022–5026.

[31] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown, "Quantitative monitoring of gene expression patterns with a complementary DNA micorarray," *Science*, vol. 270, pp. 467–470, 1995.

[32] J. L. Derisi, V. R. Iyer, and P. O. Brown, "Exploring the metabolic and genetic control of gene expression on a genomic scale," *Science*, vol. 278, no. 5338, pp. 680–686, 1997.

[33] D. Higgins and W. Taylor, *Bioinformatics: Sequence, Structure, and Databanks: A Practical Approach*. Oxford, U.K.: Oxford Univ. Press, 2000.

[34] K. J. Breslauer, R. Frank, H. Blöcker, and L. A. Marky, "Predicting DNA duplex stability from the base sequence," in *Proc. National Academy Sciences United States America*, vol. 83, 1986, pp. 3746–3750.

K. C. Tan (M'99) received the B. Eng. degree with first class honors in electronics and electrical engineering, and the Ph.D. degree from the University of Glasgow, Glasgow, U.K., in 1994 and 1997, respectively.

He was with the Center for Systems and Control and the Evolutionary Computing Group, Glasgow, Scotland, before joining the Department of Electrical and Computer Engineering, National University of Singapore, as an Assistant Professor in 1997. He has authored, or coauthored, more than 90 journal and conference publications and has served as a program committee or organizing member for many international conferences. His current research interests include computational intelligence, evolutionary computing, intelligent control, and engineering designs optimization.

Dr. Tan is Associate Editor for IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.

Arthur Tay received the B.Eng. degree with honors in electrical engineering and the Ph.D. degree in electrical engineering, in 1995 and 1998 respectively, from the National University of Singapore, Singapore.

He is a Visiting Scholar with the Information System Laboratory at Stanford University, Stanford, CT, from 1998 to 2000. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering in the National University of Singapore. His research interests include process control, application of control, optimization and signal processing in semiconductor manufacturing, microlithography, and computational intelligence.

J. Cai received the B.Eng. degree from Tsinghua University, Beijing, China in 2000 and the M.Eng. degree in Electrical and Computer Engineering from National University of Singapore in 2003.

His research interests include evolutionary algorithms, distributed computing, and artificial intelligence.