Applying genetic algorithms to decision making in autonomic computing systems

Conference Paper · January 2009

DOI: 10.1145/1555228.1555258 · Source: DBLP

David B Knoester

43 PUBLICATIONS **579** CITATIONS

SEE PROFILE

Betty H.C. Cheng
Michigan State University

282 PUBLICATIONS **8,136** CITATIONS

SEE PROFILE

Philip K. Mckinley
Michigan State University

162 PUBLICATIONS **4,307** CITATIONS

SEE PROFILE

Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems

Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng, Philip K. McKinley
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824
{ramir105, dk, chengb, mckinley}@cse.msu.edu

ABSTRACT

Increasingly, applications need to be able to self-reconfigure in response to changing requirements and environmental conditions. Autonomic computing has been proposed as a means for automating software maintenance tasks. As the complexity of adaptive and autonomic systems grows, designing and managing the set of reconfiguration rules becomes increasingly challenging and may produce inconsistencies. This paper proposes an approach to leverage genetic algorithms in the decision-making process of an autonomic system. This approach enables a system to dynamically evolve reconfiguration plans at run time in response to changing requirements and environmental conditions. A key feature of this approach is incorporating system and environmental monitoring information into the genetic algorithm such that specific changes in the environment automatically drive the evolutionary process towards new viable solutions. We have applied this genetic-algorithm based approach to the dynamic reconfiguration of a collection of remote data mirrors, with the goal of minimizing costs while maximizing data reliability and network performance, even in the presence of link failures.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*

General Terms

Experimentation.

Keywords

Autonomic computing, evolutionary algorithm, genetic algorithm, intelligent control, distributed systems

1. INTRODUCTION

As distributed computing applications grow in size and complexity in response to increasing computational needs, it is increasingly difficult to build a system that satisfies all requirements and design constraints that it will encounter during its lifetime. Many of these systems must operate continuously, disallowing periods of downtime while humans modify code and fine-tune the system. For instance, several studies [5, 27] document the severe financial penalties incurred by companies when facing problems such as data loss and data inaccessibility. As a result, it is important for applications to be able to self-reconfigure in response to changing requirements and environmental conditions [22]. IBM proposed autonomic computing as a means for automating software maintenance tasks [16]. Autonomic computing refers to any system that manages itself based on a system administrator's high-level objectives while incorporating capabilities such as self-reconfiguration and self-optimization. Typically, developers encode reconfiguration strategies at design time, and the reconfiguration tasks are influenced by anticipated future execution conditions [3, 8, 12, 30]. We propose an approach for incorporating genetic algorithms [10] as part of the decision-making process of an autonomic system. This approach enables a decision-making process to dynamically evolve reconfiguration plans at run time.

Autonomic systems typically comprise three key elements: monitoring, decision-making, and reconfiguration. *Monitoring* enables an application to be aware of its environment and detect conditions that warrant reconfiguration; *decision-making* determines which set of monitored conditions should trigger a specific reconfiguration response; and *reconfiguration* enables an application to change itself in order to fulfill its requirements. Many adaptive and autonomic systems have applied rule-based decision-making approaches to match particular events against specific reconfiguration plans [7, 12]. Others leverage utility-based decision-making approaches to address multiple reconfiguration objectives and dimensions [3, 30]. These approaches, however, enable a system to self-adapt only against scenarios that were considered at design time. Furthermore, as the complexity of adaptive logic grows, designing and managing the set of reconfiguration rules becomes unmanageable and potentially inconsistent [3]. To address these concerns, several researchers have applied evolutionary computation techniques to the design of adaptive and autonomic systems [8, 9]. Although these approaches provide a rich set of reconfigurations, most are still limited to a specific set of reconfiguration strategies.

This paper proposes an approach to incorporate genetic algorithms in the decision-making process of autonomic systems. A genetic algorithm is a stochastic search-based technique for finding solutions to optimization and search-based problems [10]. Genetic algorithms comprise a population of individuals that encode candidate solutions in the search space. Domain-specific fitness functions are used to determine the quality of each individual. Genetic algorithms use this fitness information, along with the processes of selection, crossover, and mutation, to direct the search process towards promising parts of the search space. In practice, genetic algorithms can be surprisingly fast in efficiently searching complex, highly nonlinear, and multidimensional search spaces [18]. For this work, we developed Plato, a genetic-algorithm based decision-making process that searches the vast and complex space of possible reconfigurations with the goal of evolving suitable reconfiguration plans in response to changing requirements and environmental conditions. A key benefit of Plato is that developers need not prescribe reconfiguration plans in advance to address specific situations warranting reconfiguration. Instead, Plato harnesses the power of natural selection to evolve suitable reconfiguration plans at run time.

We have applied Plato to the dynamic reconfiguration of an overlay network [2] for distributing data to a collection of remote data mirrors [11, 13]. Plato was able to evolve suitable reconfiguration plans at run time in response to changing requirements and environmental conditions. Specifically, Plato evolved overlay networks for data diffusion that balanced the competing goals of minimizing costs while maximizing data reliability and network performance, even while facing adverse conditions such as link failures. Our preliminary results suggest genetic algorithms may be integrated with decision-making processes of autonomic systems to dynamically evolve reconfiguration plans that balance competing objectives at run time.

The remainder of this paper is organized as follows. In Section 2 we present background material on remote data mirroring systems and genetic algorithms. Section 3 overviews other approaches to decision-making and dynamic configuration of networks involving genetic algorithms. Section 4 illustrates the proposed approach for integrating Plato with the decision-making process in autonomic systems. Section 5 overviews the experimental results obtained when applying Plato to the dynamic reconfiguration of an overlay network. Section 6 discusses Plato. Lastly, in Section 7 we summarize our main findings and present future directions for our work.

2. BACKGROUND

This section briefly overviews two topics fundamental to this paper. First, we present the concept of remote data mirrors and describe the challenges that complicate their designs. We then describe genetic algorithms and explain how they search for fit solutions amidst complex solution landscapes.

2.1 Remote Mirroring

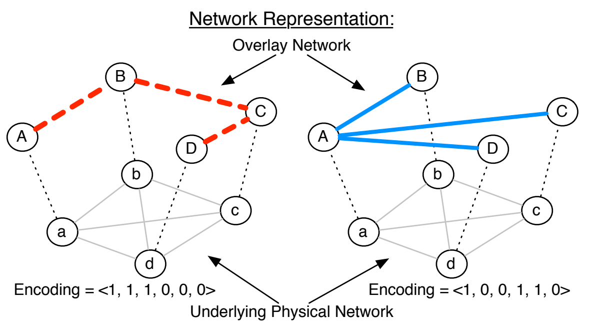
Remote data mirroring is a particular form of data mirroring in which copies of important data are stored at one or more secondary sites. The main objective of remote data mirroring is to isolate the protected data from failures that may affect the primary copy [15]. Thus, by keeping two or more copies of important information isolated from each other, access can continue if one copy is lost or becomes unreachable [11, 32]. In the event of a failure, recovery typically involves either a site failover to one of the remote data mirrors, or data reconstruction. Designing and deploying a remote mirror, however, is a complex and expensive task that should only be done whenever the cost of losing data outweighs the cost of protecting it [11]. For instance, *ad hoc* solutions may provide inadequate data protection, poor write performance, and incur high network costs [15]. Similarly, over-engineered solutions may incur expensive costs to defend against negligible risks. In general, remote data mirroring designs involve a tradeoff between better performance with lower cost against greater potential for data loss.

Two important design choices for remote data mirrors include the type of network link connecting the mirrors and the remote mirroring protocol. Each network link incurs an operational cost. In addition, each network link has a measurable throughput, latency, and loss rate that determine the overall remote mirror design performance [15]. Remote mirroring protocols, which can be categorized as either synchronous or asynchronous propagation, affect both network performance and data reliability. In *synchronous propagation* the secondary site receives and applies each write before the write completes at the primary [15]. In batched *asynchronous propagation*, updates accumulate at the primary site and are periodically propagated to the secondary site, which then applies each batch atomically. While synchronous propagation achieves zero potential data loss, it consumes large amounts of network bandwidth. Batched asynchronous propagation, on the other hand, achieves more efficient network performance than synchronous propagation, but it has a higher potential data loss.

2.2 Genetic Algorithms

Genetic algorithms are stochastic-based search techniques that comprise a population of individuals, where each individual encodes a candidate solution in a chromosome [10]. In each generation, the fitness of every individual is calculated, and a subset of individuals is selected, then recombined and/or mutated to form the next generation. For example, Figure 1 shows two individuals in a population of overlay network topologies. Each individual contains a vector to encode which overlay links are used in the corresponding topology. Specifically, given the underlying network topology vector <AB, BC, CD, AD, AC, BD>, a 1 indicates the link is part of the overlay and a 0 otherwise. Other link properties, such as the propagation methods described above, can be encoded in such vectors. In addition, each individual has an associated fitness value that is determined by evaluating the encoded solution against certain domain-specific fitness functions. This fitness information enables a genetic algorithm to choose promising solutions for further processing.

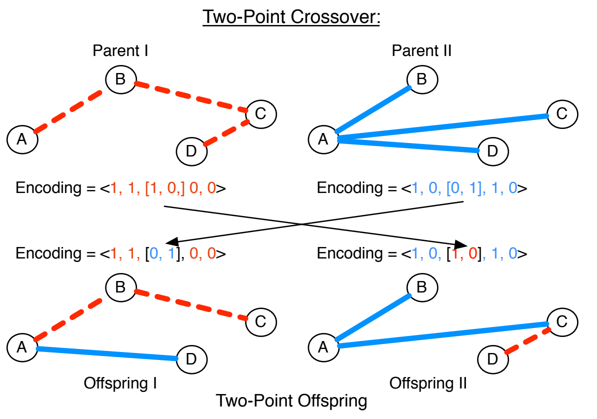
Genetic algorithms use *crossover* to exchange building blocks between two fit individuals, hopefully producing offspring with higher fitness values. The two most common forms of crossover in genetic algorithms are one-point and two-point crossover. In *one-point crossover*, a position in the chromosome is selected at random and the parts of two parents after the crossover position are exchanged. In *two-point crossover*, two positions are chosen at random and the segments between them are exchanged. Figure 2 illustrates two-point crossover for the representation described in Figure 1. Specifically, the link properties between <CD, AD> are exchanged between both parents to form two new offspring. As a result, Offspring I deactivates link <CD>,



Network Representation:
Overlay Network
B
C
A
D
b
c
a
d
Encoding = <1, 1, 1, 0, 0, 0>
Encoding = <1, 0, 0, 1, 1, 0>
Underlying Physical Network

Figure 1: Encodings of two overlay network solutions as individuals in a genetic algorithm.

from Parent I, and activates link <AD>, from Parent II. Likewise, offspring II deactivates link <AD>, from Parent II, and activates link <CD>, from Parent I.



Two-Point Crossover:
Parent I
Parent II
Encoding = <1, 1, [1, 0,] 0, 0>
Encoding = <1, 0, [0, 1], 1, 0>
Encoding = <1, 1, [0, 1], 0, 0>
Encoding = <1, 0, [1, 0], 1, 0>
Offspring I
Offspring II
Two-Point Offspring

Figure 2: Performing two-point crossover in overlay networks.

Unfortunately, genetic algorithms are susceptible to deceptive landscapes, possibly evolving suboptimal solutions (i.e., local maxima). The key idea behind *mutation* is to introduce genetic variation that may have been lost throughout the population as a result of the crossover operator [10]. Mutation takes an individual and randomly changes parts of its encoded solution based on some specified mutation rate. For example, Figure 3 illustrates a sample mutation where link <BD> is activated in one of the individuals. The activation of this link would not have been possible with the crossover operator only.

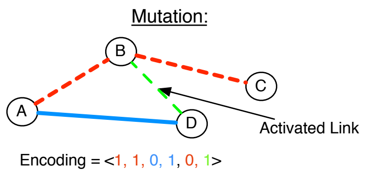


Figure 3: Performing mutation on an overlay network.

Genetic algorithms are typically executed until one of two conditions is met: Either the allocated execution time is exhausted or the algorithm converges upon a particular solution. If the execution time has been exhausted, then the best solution found thus far is considered to be the solution. Otherwise, if the algorithm has converged upon a particular solution, then the solution should be assessed for its quality. In particular, it is possible for the algorithm to converge prematurely upon a suboptimal solution. One common strategy to address this problem in practice is to reseed and restart the genetic algorithm while altering some configuration parameters such as crossover and mutation rates, forcing the search process to explore different parts of the solution landscape.

3. PROPOSED APPROACH

We designed Plato with the goal of diffusing data to a collection of remote data mirrors [11, 15] across a potentially unreliable network. Specifically, given a network containing a set of remote data mirrors, Plato must construct and maintain an overlay network so that data may be distributed to all nodes while satisfying the following properties. First, the overlay network must remain connected at all times. If the overlay network becomes disconnected for any reason, then it must be reconfigured to re-establish connectivity. Second, it should never be the case that the constructed overlay network exceeds the allocated monetary budget specified by the end-user. Third, data should be distributed as efficiently as possible, that is, minimizing the amount of bandwidth consumed when diffusing data. All considerations combined, the task of data diffusion is a multi-objective problem in which data must be distributed as efficiently as possible, while minimizing expenses and data loss.

Extensive research has been conducted on many aspects of self-adaptive and autonomic systems [16, 22, 30]. As a result, we assume the existence of a monitoring [7, 24, 25, 28] and reconfiguration [7, 26, 33] infrastructure to support self-adaptation with assurance [34, 35] at run time. The monitoring infrastructure periodically observes both the system and its execution environment and reports those values to the decision-making process. The decision-making process interprets the monitoring data, determines when a reconfiguration is required, and selects the appropriate reconfiguration plan. The reconfiguration infrastructure effects the changes throughout the system through the use of an adaptation driver. In the proposed approach, the decision-making process comprises a genetic algorithm that accepts monitoring information as input and produces reconfiguration plans as output.

3.1 Plato Design

Representation. Developers must select a suitable representation scheme for encoding candidate solutions as individuals in a genetic algorithm. For example, in Plato, every individual in the population encodes a complete network, where each link is either active or inactive and is associated with one of seven possible propagation methods (Table 1). Table 1 lists the average data batch size associated with each of the seven propagation methods, previously reported in [15]. This table also illustrates how larger amounts of data can be lost due to a failure as the amount of time for data propagation increases. In particular, synchronous propagation (time interval equal to 0) provides the maximum amount of data protection while asynchronous propagation with a 24-hour time interval provides the least amount of data protection. Using a representation scheme similar to the one previously illustrated in Figure 1, the active links encoded by each individual defines a candidate overlay network topology for data diffusion. In terms of complexity, this representation scheme comprises $2^{n(n-1)/2}$ choices for constructing an overlay network and 7 different propagation methods for each link. With 25 nodes, for example, there are $7 * 2^{300}$ possible link configurations. The total number of possible configurations makes it infeasible to exhaustively evaluate all possible configurations in a reasonable amount of time.

Table 1: Link Propagation Methods

Time Interval	Avg. Data Batch Size
0	0 GB
1 min.	0.043.6 GB
5 min.	0.2067 GB
1 hr.	2.091 GB
4 hrs.	6.595 GB
12 hrs.	15.12 GB
24 hrs.	27.388 GB

GA Operators. Crossover and mutation operators are specific to the encoding scheme used. The default crossover and mutation operators are designed to work on fixed-length binary string representations [10]. If a different representation scheme is used to encode candidate solutions, then specialized crossover and mutation operators need to be developed and used. For example, each individual in Plato encodes a candidate solution that comprises binary, integer, and floating-point values. As a result, Plato uses encoding-specific crossover and mutation operators to directly manipulate overlay network topologies. The crossover operator used by Plato (shown in Figure 2) exchanges link properties between two parents. Specifically, two network links in the link vector are selected at random and the segments between them are exchanged. Likewise, the mutation operator used by Plato (shown in Figure 3) randomly activates/deactivates a link and changes its propagation method.

GA Setup. Developers must set up the GA for the problem being solved. Typical parameters include the population size, crossover type, crossover and mutation rates, and the alloted execution time, typically expressed in generations. Table 2 gives the different parameters and values used for Plato. In particular, notice that for each generation, we perform two-point crossover on 10 individuals, thereby producing 20 offspring for the next generation. Likewise, approximately 5 individuals are mutated each generation.

Table 2: GA Parameters.

Parameter	Value
Max. population size	100
Crossover type	Two-Point.
Crossover rate	10%
Mutation rate	5%
Selection Method	Tournament (K=2)
Max generations	2500

3.2 Fitness Sub-Functions

In general, a single fitness function is not sufficient to quantify all possible effects of a particular reconfiguration plan when balancing multiple objectives [4]. Instead, developers should define a *set* of fitness sub-functions to evaluate a reconfiguration plan according to the optimization dimensions specified by end-users. Each fitness sub-function should have an associated coefficient that determines the importance or relevance of that sub-function in comparison to others. A weighted sum can be used to combine the values obtained from each fitness sub-function into one scalar value [6, 19, 23].

Plato uses several fitness sub-functions to approximate the effects of a particular overlay network in terms of costs, network performance, and data reliability. Most of the fitness sub-functions used by Plato were derived from studies on optimizing data reliability solutions [15] and modified for our specific problem. This set of sub-functions enables Plato to search for overlay networks that not only satisfy the previously mentioned properties, but that also yield the highest fitness based on how the end-user defined the sub-function coefficients to reflect the priorities for the various fitness sub-functions.

We use the following fitness sub-function to calculate an overlay network's fitness in terms of *cost*:

$$F_c = 100 - (100 * \frac{cost}{budget});$$

where *cost* is the sum of operational expenses incurred by all active links, and *budget* is an end-user supplied constraint that places an upper bound on the maximum amount of money that can be allocated for operating the overlay network. This fitness sub-function, F_c, guides the genetic algorithm toward overlay network designs that minimize operational expenses.

Likewise, we use the following two fitness sub-functions to calculate an overlay network's fitness in terms of *performance*:

$$F_{e1} = 50 - (50 * \frac{latency_{avg}}{latency_{wc}});$$

and

$$F_{e2} = 50 * (\frac{band_{sys} - band_{eff}}{band_{sys}} + bound);$$

where $latency_{avg}$ is the average latency over all active links in the overlay network, $latency_{wc}$ is the largest latency value measured over all links in the underlying network, $band_{sys}$ is the total available bandwidth across the overlay network, $band_{eff}$ is the total effective bandwidth across the overlay network after data has been coalesced, and *bound* is a limit on the best value that can be achieved throughout the network in terms of bandwidth reduction. The first fitness function, F_{e1}, accounts for the case where choosing links with lower latency will enable faster transmission rates. Likewise, the second fitness function, F_{e2}, accounts for the case where network performance can be increased by reducing the amount of data sent across a network due to write coalescing. We note that the maximum achievable value of $F_{e1} + F_{e2}$ is 100.

Lastly, we use the following two fitness sub-functions to calculate an overlay network's fitness in terms of *reliability*:

$$F_{r1} = 50 * \frac{links_{used}}{links_{max}};$$

and

$$F_{r2} = 50 * \frac{dataloss_{potential}}{dataloss_{wc}};$$

where $links_{used}$ is the total number of active links in the overlay network, $links_{max}$ is the maximum number of possible links that could be used in the overlay network given the underlying network topology, $dataloss_{potential}$ is the total amount of data that could be lost during transmission as a result of the propagation method (see Table 1), and $dataloss_{wc}$ is the amount of data that could be lost by selecting the propagation method with the largest time window for write coalescing. The first reliability fitness function, F_{r1}, accounts for the case where an overlay network with redundant links may be able to tolerate link failures while maintaining connectivity. The second reliability fitness function, F_{r2}, accounts for the case where propagation methods leave data unprotected for a period of time. We note that the maximum achievable value of $F_{r1} + F_{r2}$ is 100, the same as the fitness sub-functions for *cost* and *performance.*

The following fitness function combines the previous fitness sub-functions into one scalar value:

$$FF = \alpha_1 * F_c + \alpha_2 * (F_{e1} + F_{e2}) + \alpha_3 * (F_{r1} + F_{r2}),$$

where α_i's represent weights for each dimension of optimization as encoded into the genetic algorithm by the end user. These coefficients can be scaled to guide the genetic algorithm towards particular designs. For example, if developers want to evolve types of overlay network designs that optimize only with respect to *cost*, then α_1 could be set to 1 and α_2 and α_3 could be set to 0.

To integrate Plato into the application and factor current environmental conditions into the reconfiguration plans, developers must define a global view construct that reflects the executing system and its environment. This construct will be updated by the monitoring infrastructure and accessed by the genetic algorithm's fitness sub-functions when evaluating candidate reconfiguration plans. For instance, although Plato currently simulates the network monitoring process, each candidate overlay network in Plato stores information about the underlying complete network topology. Specifically, each link stores values such as throughput, latency, loss rate, etc. As a result, when Plato evaluates a candidate overlay network, it computes its fitness value based on current system and environmental conditions.

Rescaling Sub-Functions. If requirements are likely to change while the application executes, then developers should introduce code to *rescale* the coefficients of individual fitness sub-functions. In particular, the fitness landscape is shifted when the coefficients of a fitness sub-function are rescaled. By updating the relevance of each fitness sub-function at run time, the genetic algorithm will be capable of evolving reconfiguration plans that address changes in requirements and environmental conditions.

For example, when an overlay network link fails, Plato automatically doubles the current coefficient for network reliability. Note that although we prescribe how these coefficients should be rescaled in response to high-level monitoring events, we do not explicitly specify reconfiguration plans. That is, Plato does not prescribe how many links should be active in the overlay network, or what their propagation methods should be.

4. CASE STUDY

We conducted a case study to evolve solutions to the problem of diffusing data to a set of remote mirrors across dynamic and unreliable networks. Each of the experiments focuses on a single aspect of this problem, namely constructing and maintaining an overlay network that enables the distribution of data to all nodes. Each experiment was designed to give us insight into how different environmental factors and scenarios affect the suitability of genetic algorithms for decision-making in adaptive systems. Each experiment was simulated on a MacBook Pro with a 2.53 GHz Intel Core 2 Duo Processor and 4 GB of RAM. For each set of results presented, we performed 30 trials of the experiment and present the averaged results.

4.1 Single-dimensional optimization

The purpose of the first set of experiments is to confirm that for degenerate scenarios involving single fitness sub-functions, Plato would produce solutions consistent with those that can be predicted. As a representative example, consider the overlay network shown in Figure 4, which was evolved by Plato when only cost was considered, i.e., $\alpha_1 = 1$, $\alpha_2 = 0$, $\alpha_3 = 0$. This overlay network comprises 24 links and connects all remote data mirrors in the network. The genetic algorithm was able to reduce the overlay network to a minimum spanning tree that connects all remote data mirrors while incurring operational costs significantly below the maximum allocated budget. Figure 5 shows the maximum fitness achieved by the candidate overlay networks as Plato executed. This plot illustrates how Plato converged upon an overlay network topology with a fitness value of approximately 50, indicating that Plato found overlay networks whose operational costs were roughly 50% of the allocated budget. Although the first few hundred generations obtained negative fitness values due to ill-formed candidate topologies that were either disconnected or exceeded the allocated budget, Plato had found suitable overlay network designs by generation 500 (approximately 30 seconds on the MacBook Pro).

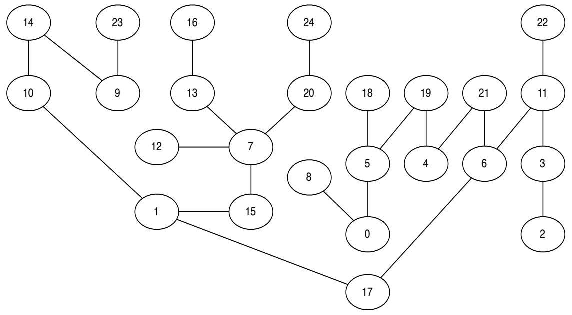


Figure 4: Overlay network produced when optimizing for cost.

For the next experiment, reliability was chosen as the single optimization criterion, i.e., $\alpha_1 = 0$, $\alpha_2 = 0$, $\alpha_3 = 1$. The evolved overlay network provides the maximum amount of reliability possible by activating all 300 links. Furthermore, the dominant propagation method for this overlay network was synchronous propagation, which minimizes the amount of data that can be lost during transmission.

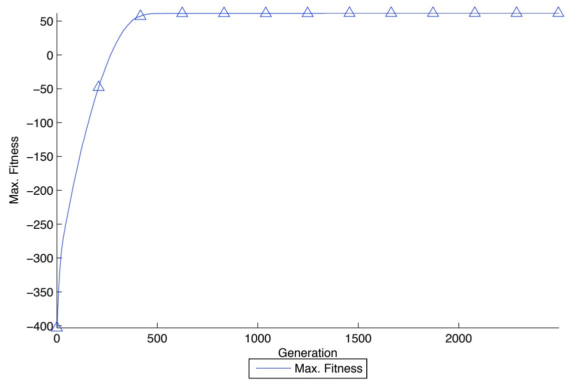


Figure 5: Fitness of overlay networks when optimizing for cost only.

Figure 6 plots the maximum fitness achieved by Plato in this experiment. Plato converged upon a maximum fitness value of 88. In the context of reliability, a value of 88 means that although the overlay network provides a high-level of data reliability, it is not completely immune against data loss. Specifically, even though all 300 links were activated in the overlay network to provide redundancy against link failures, not every link in the overlay network used a synchronous propagation method. Instead, a few links in the overlay network used asynchronous propagation methods with 1 and 5 minute time bounds. Nonetheless, we note the rapid development of fit individuals achieved by generation 600; by this point, Plato had evolved complete overlay networks with most links using synchronous propagation.

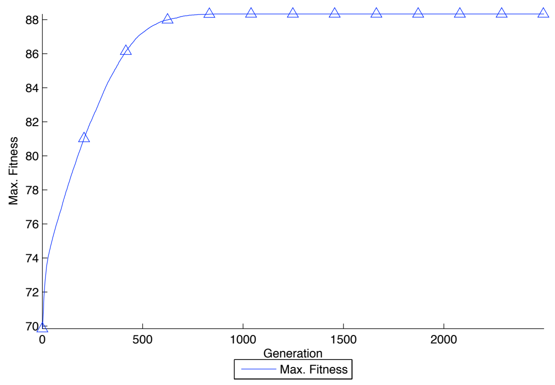


Figure 6: Fitness of overlay networks when optimizing for reliability only.

4.2 Multiple Dimensional Optimization

Next, we conducted an experiment to assess whether Plato is able to efficiently balance multiple objectives, namely, cost, performance, and reliability. For this experiment, we configured Plato to produce network designs that emphasize performance and reliability over cost, i.e., $\alpha_1 = 1$, $\alpha_2 = 2$, $\alpha_3 = 2$. Figure 7 shows a representative overlay network design that Plato evolved. This overlay network comprises 32 active links, the majority of which use asynchronous propagation methods with 1 and 5 minute time bounds. Overall, this overlay network provides a combination of performance and reliability while keeping operational expenses well below the allocated budget.

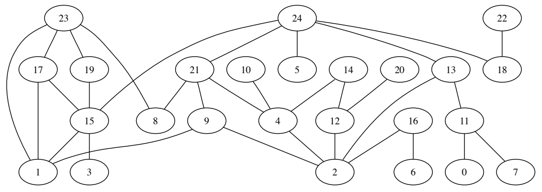


Figure 7: Overlay network produced when optimizing for cost, performance, and reliability.

Figure 8 plots the average rate at which Plato converged on the resulting overlay network designs. On average, Plato terminated within 3 minutes. In particular, note that Plato found relatively fit overlay networks by generation 500 (approximately 30 seconds). Thereafter, Plato fine-tuned the overlay network to produce increasingly more fit solutions. For instance, Figure 9 plots the average number of active links in the evolved overlay networks. At first, the more fit overlay networks were those that comprised the fewest number of active links while still maintaining connectivity. By the end of the run, 8 additional links had been added to the overlay network. Although these additional edges increased the overall operational cost of the overlay network, they also increased the network's fault tolerance against link failures, thus improving the overlay's reliability fitness value. Moreover, subsequent generations achieved higher fitness values by using asynchronous propagation methods of 5 minutes and 1 hour, thus improving network performance while providing some level of data protection as it is transmitted.

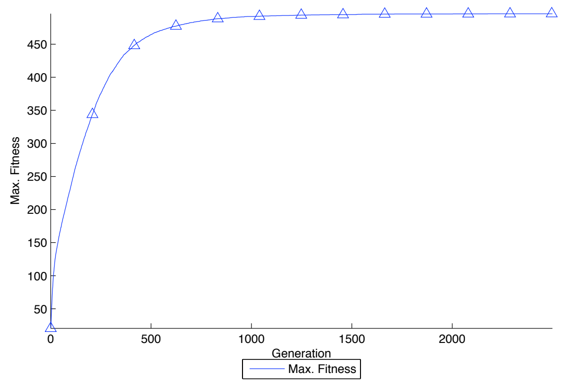


Figure 8: Maximum fitness of overlay networks when optimizing for cost, performance, and reliability.

4.3 Reconfiguration Against Link Failures

We next conducted a two-step experiment to assess the feasibility of using Plato to dynamically reconfigure the overlay network topology in real-time. First, we ran Plato to produce an initial overlay network design whose primary design objective was minimizing operational costs, i.e., $\alpha_1 =$

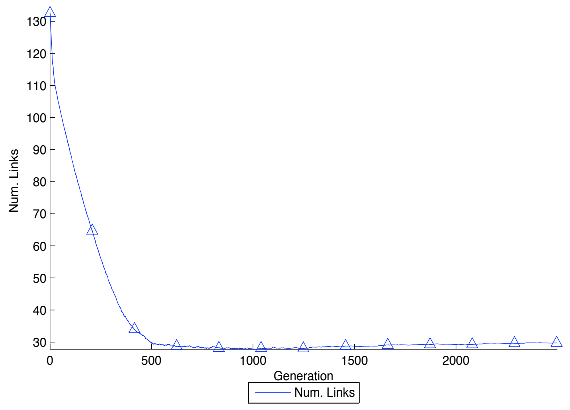


Figure 9: Number of links active in overlay network when optimizing for cost, performance, and reliability.

1, $\alpha_2 = 0$, and $\alpha_3 = 0$. Although we could have generated a design to account for cost and reliability, the objective of this experiment was to force the reconfiguration of the overlay network. Figure 10 presents a representative overlay network evolved by Plato that comprises 24 active links, a minimum spanning tree. We then selected an active link at random and set its operational status to *faulty*. This link failure disconnected the overlay network and thus prompted Plato to address this change in the underlying topology by evolving a new reconfiguration plan.

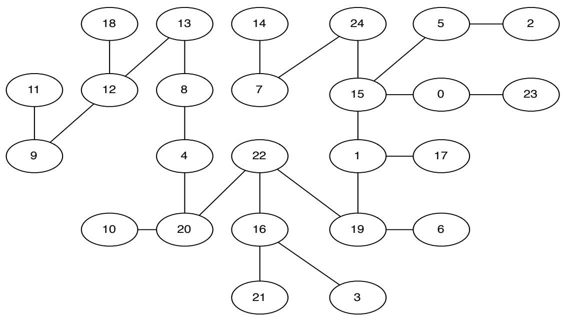


Figure 10: Initial overlay network topology with cost being the lone design factor.

In response, Plato evolved a new overlay network topology that addressed these environmental changes. Since the initial overlay network suffered from a link failure, the individual fitness sub-functions were automatically rescaled such that reliability became the primary design concern. Whenever an individual was evaluated, if the encoded overlay network made use of the faulty link, then it was severely penalized by assigning it a low fitness value. Figure 11 shows the overlay network that evolved in response to the environmental change in the underlying network. This new overlay network, with 6 redundant links, provides more reliability against link failures than the initial overlay network.

Figure 12 plots the maximum fitness achieved by Plato as it evolved both the initial and the reconfigured overlay network designs. We terminated an active link at generation 2500. As a result, the maximum fitness achieved at generation 2501 dropped to negative values. Within roughly 1000 generations (1 min.), Plato had evolved considerably more fit overlay network topologies. Notice the relative difference in maximum fitness achieved by Plato before and after reconfiguration. The initial overlay network optimizes only with respect to operational costs, while the reconfigured overlay network optimizes primarily for reliability, but also optimizes with respect to operational costs. Since Plato doubled the coefficients for reliability in comparison to cost ($\alpha_1 = 1$, $\alpha_2 = 2$, and $\alpha_3 = 2$), candidate overlay networks after generation 2500 achieved a higher relative fitness value than the initial overlay network.

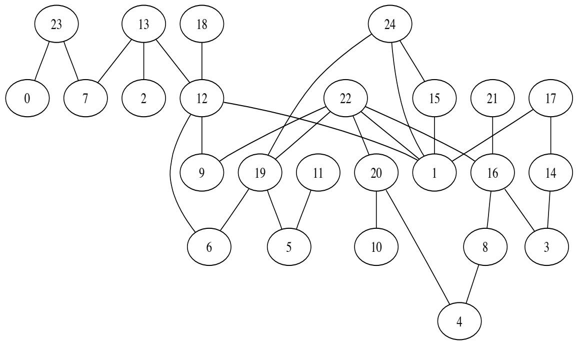


Figure 11: Overlay network topology evolved in response to a link failure.

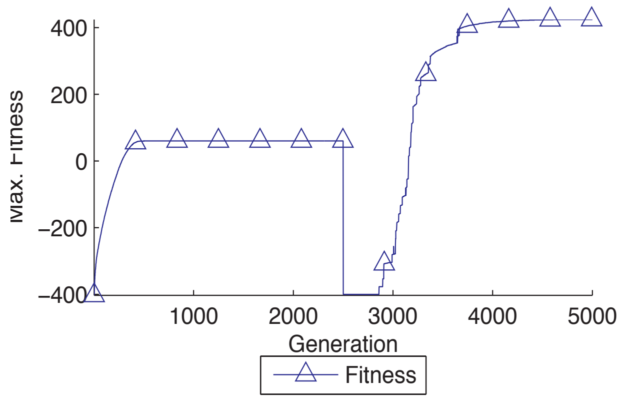


Figure 12: Maximum fitness achieved before and after reconfiguration.

Figure 13 plots the average number of active links in both the initial and reconfigured overlay network designs. While the initial overlay design obtains a higher fitness by reducing the number of active links, the reconfigured overlay design obtains a higher fitness by adding several active links to improve robustness against future link failures. Figure 14 plots the average potential data loss in both the initial and reconfigured overlay networks. The average data loss, measured on a log scale, is a byproduct of the propagation methods. Lower values of data loss imply data is better protected against link failures and vice-versa. After a link failure occurs, the reconfigured overlay network design reset *most* propagation methods to either synchronous or asynchronous propagation with a 1 minute time bound, thus improving data protection at the expense of degraded network performance.

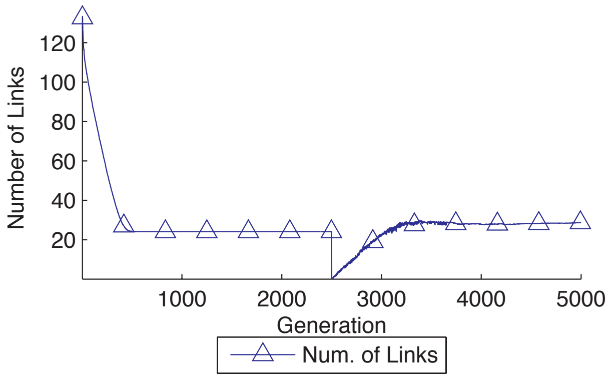


Figure 13: Number of active links in overlay network before and after reconfiguration.

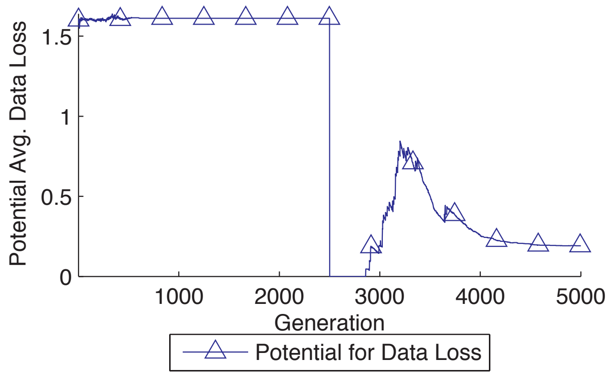


Figure 14: Potential average data loss across overlay network before after reconfiguration.

5. DISCUSSION

The experiments conducted for this paper indicate that an approach such as Plato, that incorporates genetic algorithms into decision-making processes of adaptive and autonomic systems, can be executed online to support dynamic reconfiguration. In terms of execution time, Plato terminated within 3 minutes or less, and typically converged on a solution within one minute. For an application such as remote data mirroring, we consider this time well within the acceptable range. In terms of evaluations performed, Plato typically required between $50,000$ and $100,000$ evaluations of candidate overlay network designs. Whereas other complementary approaches [17, 23] make use of simulators to assess the effects of candidate reconfiguration plans, Plato's fitness functions were mathematical calculations that were computationally inexpensive in comparison. As a result, Plato was able to perform many evaluations in a reasonably short amount of time.

Plato has several advantages over more traditional approaches for decision-making in autonomic systems. Specifically, Plato does not require developers to explicitly encode prescriptive reconfiguration strategies to address particular scenarios that may arise. Instead, Plato exploits user-defined fitness functions to *evolve* reconfiguration plans in response to changing environmental conditions. For instance, when an active link failed in our reconfiguration experiment, Plato did not explicitly encode how many or which specific links should be activated in response. Instead, Plato automatically evolved a reconfiguration plan that balanced these competing objectives at run time. This approach enables Plato to handle more reconfiguration scenarios than traditional prescriptive approaches.

One potential drawback of this approach is that genetic algorithms are not guaranteed to find the optimal solution. As a result, Plato may not find suitable solutions for some domains and problems. In these situations, however, traditional decision-making approaches could be integrated with Plato. Specifically, Plato could evolve reconfiguration plans in the background in case a reconfiguration strategy is not available for the current system conditions. In addition, Plato uses fitness functions that are not computationally expensive.

6. RELATED WORK

This section presents related work in utility-based decision-making processes, remote mirroring, and the application of genetic algorithms to the construction of dynamic overlay networks.

Utility-based Decision Making. Walsh *et al.* introduced an architecture for incorporating utility functions as part of the decision-making process of an autonomic system [30]. Utility functions were shown to be effective in handling reconfiguration decisions against multiple objectives. In the context of autonomic computing, utility functions map possible states of an entity into scalar values that quantify the desirability of a configuration as determined by user preferences. Given a utility function, the autonomic system determines the most valuable system state and the means for reaching it. In the approach proposed in [30], a utility calculator repeatedly computes the value that would be obtained from each possible configuration. Despite their advantages, utility functions may suffer from complexity issues as multiple dimensions scale depending on the evaluation method used. In contrast, although genetic algorithms use fitness functions, which are akin to utility functions, the process of natural selection efficiently guides the search process through the solution space.

Automated Design of Remote Mirrors. Researchers have developed automated provisioning tools to alleviate the high complexity associated with designing data reliability systems, such as remote mirrors and tape backups [1, 15]. These automated tools rely on mathematical solvers to produce optimal designs. As a result, these approaches require a set of functions to quantify the effects of particular configurations in terms of design costs, network performance, and data reliability. However, such approaches may not scale well as the complexity of the solution grows [14]. Furthermore, designs are produced according to expected usage rates and environmental conditions, which may change once the system is deployed. For example, the methods described in [11, 15] do not address dynamically reconfiguring the system when actual conditions differ from those assumed at design time.

Genetic Algorithms for Data Replication. Loukopoulos *et al.* [19] applied genetic algorithms to the problem of file replication in data distribution. Specifically, some files are replicated at multiple sites to reduce the delay experienced by distributed system end-users. The decision as to which files to replicate and where to replicate them is a constraint optimization problem considered to be NP-complete [23]. The initial approach in [19] leveraged a genetic algorithm to solve the file replication problem when read and write demands remained static. However, this initial approach was not applicable when read and write demands continuously changed. Loukopoulos *et al.* proposed a hybrid genetic algorithm that took as input a current copy distribution and produced a new file replication distribution using knowledge about the changing environment. This hybrid genetic algorithm is not executed at run time to dynamically solve the file replication problem, but instead incorporates possible dynamic changes into the initial design.

Genetic Algorithms for Dynamic Networks. Genetic algorithms have been used to design overlay multicast networks for data distribution [6, 20, 29, 31]. These overlay networks must balance the competing goals of diffusing data across the network as efficiently as possible while minimizing expenses. A common approach for integrating various objectives in a genetic algorithm is to use a cost function that linearly combines several objectives as a weighted sum [6, 19, 23]. Although most of these approaches [20, 29, 31] achieved rapid convergence rates while producing overlay networks that satisfied the given constraints, to our knowledge, the methods were not applied at run time to address dynamic changes in the network's environment.

Genetic Algorithms for Reconfiguration. Montana *et al.* [23] developed a genetic algorithm to reconfigure the topology and link capacities of an operational network in response to its operating conditions. Experimental results confirm that for small networks, comprising fewer than 20 nodes and fewer than 5 channels, the optimization algorithm would support online adaptation. However, due to the computational expenses, online adaptation was not supported for larger networks. Specifically, the approach in [23] made repeated use of a network simulator, ns/2 [21], to accurately model the effects of topology reconfigurations at run time. Plato, on the other hand, makes use of computationally inexpensive fitness functions to reduce the time required for convergence.

7. CONCLUSION

Having examined the evolution of reconfiguration plans by Plato, we make several observations. First, it is possible to integrate genetic algorithms with decision-making processes of autonomic systems to dynamically evolve reconfiguration plans that balance competing objectives at run time. Second, decision-making processes that leverage genetic algorithms do not require prescriptive rules to address particular scenarios warranting reconfiguration. Instead, an approach such as Plato is able to evolve reconfiguration plans to address situations that were not anticipated at design time, by incorporating system and environmental monitoring information with a genetic algorithm. Lastly, different types of reconfiguration plans can be evolved in response to changing requirements and environmental conditions by rescaling individual fitness functions. Future directions for this work include integrating the evolved reconfiguration plans with formal verification tools to ensure the reconfigured system satisfies invariant and local properties.

8. ACKNOWLEDGMENTS

We gratefully acknowledge John Wilkes for introducing us to this application problem and his insight for asking how evolutionary computation techniques might be able to address the problem. This work has been supported in part by NSF grants CCF-0541131, CNS-0551622, CCF-0750787, CNS-0751155, IIP-0700329, and CCF-0820220, Army Research Office W911NF-08-1-0495, Ford Motor Company, and a grant from Michigan State University's Quality Fund.

9. REFERENCES

[1] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions Compututing Systems*, 19(4):483–518, 2001.

[2] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *ACM SIGOPS Operating Systems Review*, (5):131–145, 2001.

[3] S.-W. Cheng, D. Garlan, and B. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 International Workshop on Self-adaptation and Self-Managing Systems*, pages 2–8, New York, NY, USA, 2006. ACM.

[4] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, 2001.

[5] EagleRock2001. Online survey results: 2001 cost of downtime. Eagle Rock Alliance Ltd, http://contingencyplanningresearch.com/2001 Survey.pdf, August 2001.

[6] R. Fabregat, yezid Donoso, B. Baran, F. Solano, and J. L. Marzo. Multi-objective optimization scheme for multicast flows: A survey, a model and a MOEA solution. In *Proceedings of the 3rd International IFIP/ACM Latin American Conference on Networking*, pages 73–86, New York, NY, USA, 2005. ACM.

[7] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[8] H. J. Goldsby and Betty H.C. Cheng. Automatically generating behavioral models of adaptive systems to address uncertainty. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, pages 568–583, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] H. J. Goldsby, Betty H.C. Cheng, P. K. McKinley, D. B. Knoester, and C. A. Ofria. Digital evolution of behavioral models for autonomic systems. In *Proceedings of the fifth IEEE International Conference on Autonomic Computing*, pages 87–96, Washington, DC, USA, 2008. IEEE Computer Society.

[10] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, USA, 1992.

[11] M. Ji, A. Veitch, and J. Wilkes. Seneca: Remote mirroring done write. In *USENIX 2003 Annual*

Technical Conference, pages 253–268, Berkeley, CA, USA, June 2003. USENIX Association.

[12] G. Kaiser, P. Gross, G. Kc, and J. Parekh. An approach to autonomizing legacy systems. In *Proceedings of the first Workshop on Self-Healing, Adaptive, and Self-MANaged Systems*, 2002.

[13] K. Keeton, D. Beyer, E. Brau, and A. Merchant. On the road to recovery: Restoring data after disasters. *SIGOPS Operating Systems Review*, 40(4):235–248, April 2006.

[14] K. Keeton and A. Merchant. Challenges in managing dependable data systems. *SIGMETRICS Performance Evaluation Review*, 33(4):4–10, 2006.

[15] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes. Designing for disasters. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 59–62, Berkeley, CA, USA, 2004. USENIX Association.

[16] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[17] R. Khanna, H. Liu, and H.-H. Chen. Dynamic optimization of secure mobile sensor networks: A genetic algorithm. *Proceedings of the IEEE International Conference on Communications*, pages 3413–3418, June 2007.

[18] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, December 1992.

[19] T. Loukopoulos and I. Ahmad. Static and adaptive distributed data replication using genetic algorithms. *Journal Parallel Distributed Computing*, 64(11):1270–1285, 2004.

[20] J. Lu and W. Cheng. A genetic-algorithm-based routing optimization scheme for overlay network. In *Proceedings of the 3rd International Conference on Natural Computation*, pages 421–425, Washington, DC, USA, 2007. IEEE Computer Society Press.

[21] S. McCanne and S. Floyd. The lbnl network simulator. Software on-line: http://www.isi.edu/nsnam, 1997.

[22] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and Betty H.C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[23] D. Montana, T. Hussain, and T. Saxena. Adaptive reconfiguration of data networks using genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1141–1149, San Francisco, CA, USA, 2002.

[24] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cistoiu. MonALISA: A Distributed Monitoring Service Architecture. In *Proceedings of the 2003 Conference for Computing in High Energy and Nuclear Physics*, March 2003.

[25] A. J. Ramirez. Design patterns for developing dynamically adaptive systems. Master's thesis, Michigan State University, East Lansing, MI 48823, 2008.

[26] S. M. Sadjadi and P. K. McKinley. ACT: An adaptive CORBA template to support unanticipated adaptation. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 74–83, 2004.

[27] SEC2002. Summary of "lessons learned" from events of september 11 and implications for business continuity. http://www.sec.gov/divisions/marketreg/lessonslearned.htm, February 2002.

[28] C. Tang and P. K. McKinley. A distributed approach to topology-aware overlay path monitoring. In *Proceedings of the 24th International Conference on Distributed Computing*, pages 122–131, Washington, DC, USA, 2004. IEEE Computer Society.

[29] S.-Y. Tseng, Y.-M. Huang, and C.-C. Lin. Genetic algorithm for delay- and degree-constrained multimedia broadcasting on overlay networks. *Computer Communications*, 29(17):3625–3632, 2006.

[30] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of the First IEEE International Conference on Autonomic Computing*, pages 70–77, Washington, DC, USA, 2004. IEEE Computer Society.

[31] D. Wang, J. Gan, and D. Wang. Heuristic genetic algorithm for multicast overlay network link selection. In *Proceedings of the Second International Conference on Genetic and Evolutionary Computing*, pages 38–41, September 2008.

[32] R. Witty and D. Scott. Disaster recovery plans and systems are essential. Technical Report FT-14-5021, Gartner Research, September 2001.

[33] Z. Yang, Betty H.C. Cheng, R. E. Kurt Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley. An aspect-oriented approach to dynamic adaptation. In *Proceedings of the First Workshop on Self-Healing Systems*, pages 85–92, New York, NY, USA, 2002. ACM.

[34] J. Zhang and Betty H.C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering*, pages 371–380, New York, NY, USA, 2006. ACM.

[35] J. Zhang, H. J. Goldsby, and Betty H.C. Cheng. Modular verification of dynamically adaptive systems. In *Proceedings of the Eighth International Conference on Aspect-Oriented Software Development*, 2009.