Extremal Search of Decision Policies for Scalable Distributed Applications

Gang Chen, Chor Ping Low, and Zhonghua Yang

Information Communication Institute of Singapore
School of Electrical and Electronic Engineering
Nanyang Technological University, Singapore 629798

ChenGang@ntu.edu.sg

ABSTRACT

The ongoing trend of constructing open, scalable distributed systems such as peer-to-peer (P2P) systems demands for effective tools to manage the interactions between constituent entities (or nodes). One such tool is through imposing decision policies at the network level. However very few techniques are available to allow computers autonomously identify good policies with limited human intervention. In this paper, we propose an Extremal Programming (EP) algorithm to achieve automatic policy identification. The algorithm is inspired by recent advances in understanding far from equilibrium phenomena in terms of self-organized criticality (SOC). The effectiveness of EP is evaluated through a P2P application called location-aware video streaming (LAVS). The simulation studies in LAVS demonstrate that with EP, the fast and effective sharing of video streams is achieved.

Categories and Subject Descriptors

I.2.8 [**computing methodologies**]: artificial intelligence—*problem solving, control methods, and search*—Heuristic methods

1. INTRODUCTION

In recent years, through the introduction of the music-sharing application, called Napster [4], a new *"peer-to-peer"* paradigm has greatly influenced our way of constructing distributed systems over the Internet. Instead of browsing the web and trading email, ever more powerful machines at home and in the office are now connecting to each other directly, collaborating to become virtual super-computers and pervasive information warehouse [12]. The use of *decision policies* (i.e. social norms) in P2P systems has attracted increasing research interests recently [16, 19]. A *policy* is defined as a globally-recognized rule that stipulates the conditions for nodes to perform certain activities in a distributed environment. A node must make itself fully aware of the policies available in the system. When it decides to conduct certain activities with other nodes, this decision must be supported by some policies.

Nevertheless, very few computational techniques are available to allow peers autonomously identify good policies with limited human intervention. In this paper, we propsoe an *Extremal Programming* (EP) algorithm in order to improve the autonomy of the P2P network. The design of our algorithm is inspired by a recently introduced general-purpose optimization technique termed *Extremal Optimization* (EO) [3]. The derivation of the EP algorithm starts from the *extremal selection mechanism* that underlies the EO algorithm. A population-based search strategy is utilized. Each individual in the population stands for a separate policy. Every time policies with relatively low fitness values are selected and replaced by policies obtained from the crossover and local search operators. We narrowed down our focus on a simple but practically useful policy representation. Methods are proposed to evaluate the fitness of each policy in the population. Operators to perform the crossover and the local search operations over policies are also defined. The result of our design decisions is the EP algorithm, which is able to conduct policy search through evolving a population of policies.

The EP algorithm is designed for general use in mind, and its application and exploration are illustrated using a P2P networks. Specifically, policies are utilized to manage the sharing of video streams over a network of mobile phones. The EP algorithm is applied to search policies that will result in fast and efficient sharing of video streams. Simulation studies have been performed and we found that the EP algorithm is effective in our P2P applications.

The remainder of this paper is organized as follows. Section 2 briefly compares related research. Section 3 presents the derivation of the extremal programming algorithm (EP). Section 4 describes a location-aware video streaming system and shows the simulation results after utilizing the EP algorithm. Finally Section 5 concludes this paper.

2. RELATED WORK

The extremal programming (EP) algorithm proposed in this paper is inspired by a recently introduced optimization technique termed Extremal Optimization (EO) [2, 3]. EO is a

INFOSCALE 2007, June 6-8, Suzhou, China

DOI 10.4108/infoscale.2007.207

general-purpose local search heuristic based on recent progress in understanding far-from-equilibrium phenomena in terms of self-organized criticality (SOC) [13]. In the primitive form of EO, a solution ρ to a problem is comprised of multiple constituent variables x_i, $\rho = \{x_1, x_2, ..., x_n\}$. Each variable x_i is associated with a fitness λ_i. The solution to be used in successive search steps in EO is obtained by changing the variable x_i that has the lowest λ_i among all variables. Research shows that EO exhibits large fluctuations in dynamics, allowing it to efficiently explore the solution space. Meanwhile, the *extremal selection mechanism* enforces frequent returns to near-optimal solutions.

Every implementation of EO demands a method for evaluating the fitness of each variable that together constitutes a solution to a problem. For many complex problems, it might not be easy to efficiently estimate the impact of these variables (as variables depend upon each other in a complex manner), which is essential to the success of EO. The EP algorithm in this paper is built on top of the extremal selection mechanism to facilitate effective exploration of a policy space. However, a population-based search strategy instead of the local search adopted in EO is utilized to circumvent this fitness evaluation problem. EP is designed to evolve a population of solutions (or partial solutions) to a problem. As edvidenced by many evolutionary algorithms such as the genetic algorithms (GA), it was often easy to evaluate the performance (i.e. fitness) of a solution as a whole than evaluating each constituent variable. In an attempt to further enhance the search effectiveness, a crossover operator as well as a local search operator are employed when replacing low-fitness solutions.

Similar to GA and EO, EP is an algorithm motivated by evolution. Among the various evolutionary algorithms, it is straightforward to show the functional similarity between EP and the Genetic Programming (GP) algorithm [8, 9]. GP can be considered as a hierarchical genetic algorithm operating on populations of computer programs. It aims at efficient search of computer programs for solving many general problems, such as the decision-making problems, by using Darwinian principles of reproduction and survival of the fittest. Each candidate program in GP is typically represented in terms of a tree structure, with the nodes of the tree denoting operators, variables, or constants. While GP takes a hierarchical representation of a computer program for decision-making problems, the EP algorithm takes a rather plain view. The policies to reach a decision are evolved as a population with low-fitness policies replaced by newly-generated policies. The decision-making process is driven by the combination of policies in the population, instead of any single candidate program. Based on our knowledge, there are few techniques of using GP to conduct policy search. One exception is an evolutionary policy iteration algorithm for solving Markov Decision Processes (MDP) [6]. However, the result of this algorithm is a single policy instead of a group of policies. Based on this understanding, we see that EP is at least a good complement to GP, especially when decision policies rather than a computer program are more desirable.

In the literature, machine learning techniques, such as the reinforcement learning algorithms [18, 20], have been successfully exploited to find decision rules for many problems. As Koza pointed out [9], reinforcement learning, in general, requires that a discounted estimate of the expected future payoffs be calculated for each system state. Due to the problem of the *curse of dimensionality* [1], a large number of trials are to be made over a large number of combinations of possibilities before acceptable decision rules can form. While reinforcement learning stands for a more extensive search of the policy space, the EP algorithm may serve as a more efficient search strategy even without relying on any system state information. It is interesting to see, however, the potential of combining these two types of algorithms in order to exploit the advantage of both.

For some applications, it is possible to prepare a group of decision-making samples based on human experts' knowledge, practical principles, or system experiment data. Very often useful decision policies can be discovered from these samples in the form of decision trees [14], decision rules [10], or even fuzzy rule systems [7]. The usefulness is judged according to the requirement that the identified policies should (1) accurately extract the essential information (or patterns) hidden behind the samples, and (2) precisely predict desirable decisions in situations not covered by these samples. Many techniques such as ID3, C4.5 [15], EDRL-MD [10] have been proposed to achieve these goals. Industrial applications in fields like system monitoring and control further bring these techniques to fruition. The EP algorithm in this paper, however, addresses a different issue when samples required by these techniques are unavailable.

In this paper, the EP algorithm is applied to improve the performance of a Peer-to-Peer (P2P) network. Traditionally, P2P research focuses mainly on the problems of scalable data lookup and effective data sharing [17, 21]. Depending on the way they are connected and how the data they contain is arranged, P2P networks have been classified into two categories, namely, the structured networks and the un-structured networks [11]. Although P2P file sharing has won an astounding popularity, there is a continued interest of exploring other exciting P2P applications in both structured and un-structured networks. One such application over a network of mobile phones will be considered in this paper. The protocol that lays the foundation of our P2P network shares many similarities to a protocol proposed by Chakravarti *et. al.* for organizing computation tasks in P2P networks [5]. Nevertheless the object of our protocol is to support policy-driven decision making in the process of sharing video streams originated from certain geographic locations.

3. THE EXTREMAL PROGRAMMING ALGORITHM

The EP algorithm pre-assumes that the decision-making process of autonomous entities (e.g. a node in a P2P network) is driven by their policies. In order to understand the role played by these policies, a simple policy representation will be introduced first. Similar with a decision rule, a policy ϱ is comprised of two parts, namely, the *condition part* and the *decision part*. The decision part specifies the type of decisions guarded by the policy ϱ (e.g. upload data to other nodes in a P2P network). Potentially, there are several *alternatives* α involved in this decision. For a box

moving robot, the alternatives might be the directions to move the box [9]. While in a P2P network, the alternatives could be the set of nodes to upload data. Policy ϱ actually *controls* the alternatives acceptable by a decision.

A policy ϱ differentiates multiple alternatives α through specifying a group of *decision criteria* in its condition part. In the context of a P2P network, a criterion corresponds to certain characteristic of a node under consideration, such as the available storage space of the node. In case when multiple nodes are being considered for uploading data to them, a policy may stipulate that only those nodes with relatively large storage space are selected to receive the data. To formalize this description, a criterion ψ_i is defined as a two-element tuple $< \mu(\cdot), \theta(\cdot) >$. $\mu(\cdot)$ is a *characteristic function*. It takes as its input an alternative α and returns a real value that measures certain characteristic of α. $\theta(\cdot)$ is a *response function*. The input of $\theta(\cdot)$ is a vector of real numbers. The output gives a response value to each member of the input vector. Specifically, $\theta_{x_i}((x_1, ..., x_n))$ denotes the response value of x_i when the input is the vector $(x_1, ..., x_n)$, $\theta_{x_i}((x_1, ..., x_n)) \in [0, 1]$. Upon considering a set of alternatives $\{\alpha_1, ..., \alpha_n\}$, an alternative α_i is said to be *acceptable* with criteria $\psi =< \mu(\cdot), \theta(\cdot) >$ if and only if

1. For all $\alpha_i \in \{\alpha_i, ..., \alpha_n\}$, $\mu(\alpha_i) = x_i$. And
2. $\theta_{x_i}((x_1, ..., x_n)) > \epsilon$

ϵ is a *decision threshold* between 0 and 1. In our policy implementation, $\epsilon = 0.5$. Suppose that a policy ϱ has specified n criteria $\{\psi_1, ..., \psi_n\}$ in its condition part. Among a set of alternatives $\{\alpha_1, ..., \alpha_m\}$, an alternative α_i is said to be *acceptable* with the policy ϱ iff it is acceptable with all criteria ψ_i in ϱ. For this reason, the condition part of the policy ϱ can be denoted as

$$\psi_1 \wedge \psi_2 \wedge ... \wedge \psi_n$$

It is highly possible that multiple policies will be applied to drive the same type of decision (i.e. policies have identical decision parts). However, due to their varied condition parts, these policies may not be semantically *consistent*. In order to solve this problem of inconsistency, a priority order is imposed over policies. Formally, let $\mathbf{P}$ denote a set of k decision policies. Each policy ϱ_i in $\mathbf{P}$ is associated with a real-valued priority $\pi(\varrho_i)$. All the policies in $\mathbf{P}$ form a *priority list: List* $=< \varrho_1, ..., \varrho_k >$, ordered decreasingly based on their priority values. Upon making a decision that involves a set of alternatives $\mathbf{A} = \{\alpha_1, ..., \alpha_n\}$, the policies in $\mathbf{P}$ are applied one by one as follows. The first policy ϱ_1 in the priority list is applied to $\mathbf{A}$. If any alternative α_i is acceptable by ϱ_1, the decsision to choose α_i is then made, and the decision-making process terminates. When multiple alternatives are acceptable by ϱ_1, the decision is made to choose either all of them or a randomly selected alternative (depending on domain-specific settings). If no alternative is acceptable by ϱ_1, ϱ_2 is applied to $\mathbf{A}$. This process continues until either an acceptable alternative has been identified or policies in $\mathbf{P}$ have all been applied. The objective of the EP algorithm is to identify a population of policies with the above representation. The major steps of EP is shown in Figure 1.

1. Randomly initialize a population of policies $\{\varrho_1, \varrho_2, ..., \varrho_m\}$.
2. For the current population of policies:
   (a) Determine the priority order of each policy, $\pi(\varrho_i)$.
   (b) Evaluate the fitness of each policy, $\lambda(\varrho_i)$.
   (c) Select a policy ϱ_i within the population.
   (d) Generate a set of candidate policies $G(\varrho_i, N(\varrho_i))$ and evaluate their fitness.
   (e) Replace ϱ_i and $N(\varrho_i)$ with two policies in $G(\varrho_i, N(\varrho_i))$.
3. Repeat at step 2 as long as desired.
4. Return the policy population finally reached.

Figure 1: The Extremal Programming Algorithm.

In addition to the methods for fitness evaluation, candidate policy generation and replacement, the EP algorithm demands for a method to evaluate the priority order of each policy (Ref. step 2a in Figure 1), which will be discussed in the following subsections. Now it is necessary to define the *neighbor relation* concept. In EP all policies in the population are linked together to form a ring. As policies may have different decision parts, the neighbor of any policy ϱ, $N(\varrho)$, should satisfy either of two conditions:

1. Policy $N(\varrho)$ has the same decision part as policy ϱ.
2. On the route from policy ϱ to policy $N(\varrho)$ by following the directed ring link, no other policies have the same decision part as policy ϱ.

The rational of this definition of the neighbor relation is to remove the possibility of generating candidate policies from two policies that have different decision parts. The system performance concept is introduced at step 4 of Figure 1. In the ES algorithm, the system performance refers to some globally-defined real-valued performance index, which is expected to be improved through changing the decision policies. To be more general, we consider a system as a black box that accepts a population of policies and in return produces a real value to indicate its performance.

3.1 Determine the Priority of Each Policy

For any given set of policies $\mathbf{P}$, through changing the priority order, the system may produce considerably different performance results. Our goal is to find a priority order over $\mathbf{P}$ such that the system can achieve a relatively better performance. As the effect of these policies is not known *a priori*, a simple heuristic as shown in Figure 2 is used to determine the priority order.

1. Randomly assign a priority value to each policy ϱ in the population.
2. Repeat for l iterations:
   (a) Order the policies according to their priority values to form a *priority list* $List =< \varrho_1, ..., \varrho_m >$.
   (b) Repeat for each policy ϱ_i in the population:
      i. Move ϱ_i to the first position in the priority list $List$.
      ii. Evaluate the system performance Per_i and set the priority value of ϱ_i to Per_i.
      iii. Move ϱ_i back to its original position in the list $List$.

Figure 2: Priority value determination method.

3.2 Evaluate the Fitness of Each Policy

As the ultimate goal is to find a group of policies so as to improve the system performance, it is desirable to build a strong correlation between the fitness of policies and the system performance. Specifically, the fitness of a policy ϱ_i should be relatively higher if the system performance becomes worse after removing this policy from the population (i.e. *positive impact*). Conversely, $\lambda(\varrho_i)$ should be relatively lower if the system performance can even be improved without this policy (i.e. *negative impact*). Figure 3 shows the method utilized by the EP algorithm for evaluating the fitness of policies. In essence, this method intends to estimate the impact of each policy ϱ_i through calculating the difference between the system performance obtained with and without policy ϱ_i. This estimated impact will serve as the fitness of ϱ_i.

1. Record the system performance Per observed when policies in the population are ordered according to $List =< \varrho_1, ..., \varrho_m >$.
2. Repeat for each policy ϱ_i in the population:
   (a) Remove ϱ_i from the policy population.
   (b) Evaluate the system performance Per_i.
   (c) Set $\lambda(\varrho_i) = Per - Per_i$.
   (d) Add ϱ_i back to the policy population.

Figure 3: Policy fitness evaluation method.

3.3 Candidate Policy Generation

According to Figure 1, a set of candidate policies are to be generated to replace selected policies ϱ and $N(\varrho)$. As the decision parts of ϱ and $N(\varrho)$ are identical, the crossover as well as the local search operator will only be applied to the condition parts of the two policies. Suppose that a *total-order relation* $\leq_\psi$ is defined over the set of decision criteria that are possible to be employed by a policy. A uniform crossover operator as shown in Figure 4 is applied for generating candidate policies.

1. Order the criteria in policies ϱ and $N(\varrho)$ based on $\leq_\psi$ to form two lists, $List$ and $List'$, respectively. Let ψ_i stand for the i-th element of $List$. Let ψ'_j stand for the j-th element of $List'$.
2. Create two policies ϱ_1 and ϱ_2 with initially empty condition part.
3. Initialize two integers $a = 1$, $b = 1$.
4. If $\psi_a \leq_\psi \psi'_b$ or the end of $List'$ is reached:
   (a) ψ_a is assigned to eigher ϱ_1 or ϱ_2 with equal probability, provided that no criteria ψ^* in the selected policy satisfies $\psi_a \leq_\psi \psi^*$.
   (b) a is incremented by 1.
5. if $\psi'_b \leq_\psi \psi_a$ or the end of $List$ is reached:
   (a) ψ'_b is assigned to either ϱ_1 or ϱ_2 with equal probability, provided that no criteria ψ^* in the selected policy satisfies $\psi'_b \leq_\psi \psi^*$.
   (b) b is incremented by 1.
6. Repeat at step 4 util both $List$ and $List'$ reach their ends.

Figure 4: The uniform crossover operation in the EP algorithm.

The candidate policy set contains not only policies ϱ_1 and ϱ_2 generated in Figure 4, but also policies obtained through the local search operator. For every criterion $\psi =< \mu(\cdot), \theta(\cdot) >$ in either ϱ_1 or ϱ_2, the local search operator will introduce a small deviation to the response function $\theta(\cdot)$ (i.e. by changing parameters of the function). All candidate policies will be temporarily added to the policy population to estimate their priority and fitness. Based on the fitness evaluated, the policy replacement method is performed. The EP algorithm continues with the next iteration at step 2 in Figure 1 until a group of desirable policies have been found.

4. LOCATION-AWARE VIDEO STREAMING IN P2P NETWORKS

In this Section, a P2P application termed *location-aware video streaming* (LAVS) is explored. In LAVS, a group of nodes (i.e. mobile phones) need to share the video stream originated from one node at certain geographic location. We use this application to evaluate the effectiveness of the EP algorithm, and it is not our interest to investigate the practicality of LAVS based on the actual usage of mobile phones. Instead we assume that:

A1: Mobile phones have GPS function. Upon sharing a video stream, a phone will encode its location information into the shared video stream so that other phones are able to check whether they need to download the video stream.

A2: Initially, a mobile phone η is only aware of those phones (i.e. *neighbor nodes*) geographically close to it. η ex-

plores the mobile phone world by sending a message containing its contact information to one of its neighbors. This message is relayed between phones. When another phone η' not known to η before received the message, η' can directly contact η using the contacting information in the message. The communication bandwidth between η and η' is independent of the geographic distance between the two phones.

LAVS can be useful in many situations. For example, in case of a traffic jam caused by a transportation accident, a person who is close to the accident location can capture the scene using his mobile phone and share the video stream so that other people may know the updated progress of the police work. Based on assumptions A1 and A2, it is convenient to form a decision-making problem, which is called the LAVS problem in the sequel. Without losing generality, assume that nodes in a mobile phone network are arranged geographically based on a ring topology. Figure 5 illustrates one such ring structure.

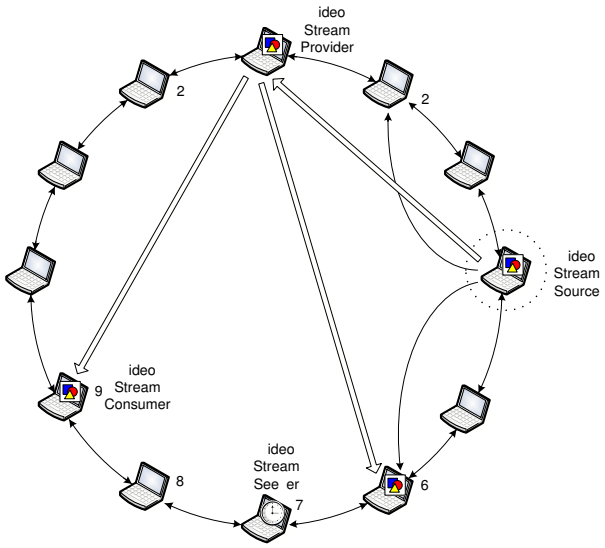

Figure 5: The location-aware video streaming (LAVS) problem.

As shown in Figure 5, each node η on the ring is assigned a positive-integer index $Ind(\eta)$ to represent its geographical location. To simplify our discussion, a node with index i is denoted as η_i, $Ind(\eta_i) = i$. Starting from the node with the lowest index, the index of each node visited by following the clockwise ring link forms an increasing arithmetic series (i.e. 1,2,3,...). Based on node indexes, the geographical distance between any two nodes η_i and η_j on the ring is defined as

$$\|\eta_i, \eta_j\| = Min \left(\begin{array}{c} |i-j|\,, \\ Min(i,j) + \|\Sigma\| - Max(i,j) \end{array} \right) \quad (1)$$

where Σ refers to the set of all nodes. $\|\Sigma\|$ returns the cardinality of Σ. With this distance concept, any node η_i initially is only aware of those nodes η_j such that $\|\eta_i, \eta_j\| \leq \kappa$ (Ref. assumption A2). η_j is called the *neighbor node* of η_i. In our simulation system, κ is set to 2. For example, the neighbors of η_4 in Figure 5 include nodes η_2, η_3, η_5, and η_6.

Nodes will generally play different roles in the process of sharing a video stream. Due to assumption A1, the sharing of multiple video streams can be differentiated and considered separately. For this reason, the LAVS problem only focuses on sharing a single stream. In Figure 5, the shared video stream is captured originally by node η_4, which is termed the *stream source*. Node η_4 uploads this video stream to node η_1. As η_1 will further upload the video stream to nodes η_6 and η_9, η_1 is termed a *stream provider*, whereas η_6 and η_9 are termed *stream consumers*. Node η_7 intends to download the video stream from η_4 but has not connected to the streaming service yet. For this reason, η_7 is called a *stream seeker*. Of course, there also exist some nodes (called *linkers*) that do not want to download video streams. Their major activity in the LAVS problem is to relay messages sent by other nodes. Starting from the stream source η_4, a *streaming tree* is formed over all nodes that actually shared the video stream. The number of child nodes of every node η in the tree (i.e. $N_c(\eta)$) is upper bounded by the *capability* of η, $C(\eta)$. The LAVS problem can be described as follows. Suppose that a P2P network as shown in Figure 5 contains only a single stream source at the beginning. Other nodes in the network are either stream seekers or linkers. The question is what local decisions should every node η make over time such that

R3: All stream seekers will become either stream providers or consumers after a short period of time.

R4: The average length of the paths from the stream source to any node in the tree is as small as possible.

R4 is desirable as uploading video streams to other nodes in the network will induce a delay for the transferred video content. It is easy to transform R3 and R4 together into a real-valued performance index J . Let $Anc(\eta)$ denote the set of ancestor nodes of any node η in a streaming tree. Moreover, Let Σ_{tr} represent the set of all nodes in the streaming tree. The performance index J is defined on a P2P network as

$$J = -\sum_{\eta \in \Sigma_{tr}} (\|Anc(\eta)\|) - \|\Sigma\| \times \|\Sigma_s\| \quad (2)$$

where Σ_s denotes the set of service seeker nodes in the network. Each node η in the network can only send *probing messages* to other nodes. Inside a probing message, node η indicates its contact information as well as its role played while sharing the video stream (e.g. stream provider or stream seeker). Let η be the node whose probing message has been received by another node η'. Node η' is possible to make three types of decisions subject to its role:

D1: Send the probing message received from η to other nodes in the network.

D2: Send the probing message of node η' itself to other nodes in the network.

D3: Replace one of the child nodes of η' in the streaming tree by node η. The replaced node will be put at the position previous occupied by η.

Each node η in the network maintains the local information of several other nodes, which will serve as the alternatives (Ref. Section 3) when η makes its local decisions. Potential alternatives include the neighbor nodes, ancestor nodes, and child nodes of η. A *video stream sharing protocol* is provided in Figure 6. Every node η will follow this protocol while making its deicisons.

1. If node η is a *stream seeker*:
   (a) Perform decision D2 with an alternative node.
2. If η is a *stream consumer* or *stream provider*, perform decision D2 with an alternative node.
3. Repeat for every probing message received from another node η':
   (a) If η is a *linker* or *stream seeker*, perform decision D1 with an alternative node.
   (b) Else if η' is a *stream seeker*:
      i. If $N_c(\eta) < C(\eta)$, set η' as a child node of η.
      ii. Else perform decision D1 with an alternative node.
   (c) Else:
      i. If $N_c(\eta) < C(\eta)$, set η' as a child node of η.
      ii. Else perform decision D3 with an alternative node.
4. Repeat at step 2.

Figure 6: The stream sharing protocol.

A policy-driven approach is utilized to guide the local decision-making of each node. In LAVS policies will be applied for selecting alternative nodes with respect to decisions D1, D2, and D3. An alternative node η is acceptable by a policy ϱ iff η is acceptable with all the criteria specified in the condition part of ϱ. The ultimate goal of the LAVS problem is therefore to find a group of policies so as to effectively improve the performance index J. The EP algorithm (Ref. Section 3) is utilized in the LAVS problem to explore the policy space. The main task of building a policy is to specify the decision criteria $\psi =< \mu(\cdot), \theta(\cdot) >$ in its condition part. Table 1 explains the meaning of those criteria that are allowable in a policy.

A simulated P2P system has been built up in order to estimate the performance index J with respect to a population of policies. In our experiment, 300 nodes are simulated under a ring topology (Ref. Figure 5). The node with index 150 is set to be the *stream source*. Other nodes are either *stream seekers* or *linkers*. The capability of each node follows a normal distribution with mean 5 and deviation 1. The

Decision type	Allowed decision criteria ψ
D1	ψ_1: The distance between an alternative node and the stream source. ψ_2: The maximum length among the paths between an alternative node and any of its descendent nodes in the streaming tree.
D2	ψ_2 ψ_3: The capability of an alternative node η', $C(\eta')$. ψ_4: $\|Anc(\eta')\|$ of an alternative node η'. ψ_5: The avaialble capability of an alternative node η', $C(\eta') - N_c(\eta')$.
D3	ψ_2, ψ_3, and ψ_5.

Table 1: Allowed criteria in a policy for each type of decisions.

simulation advances in time steps of 1 second. During each time step, every node will perform one round of the stream sharing protocol (Ref. Figure 6). The simulation terminates after 15 simulated seconds. After that, the performance index J is evaluated. J will be used to determine the priority as well as the fitness of each policy in the population and therefore drives the extremal search process. Figure 7 shows the best J found by the EP algorithm from search steps 0 to 200 when about 60% of the 300 nodes are initially stream seekers.

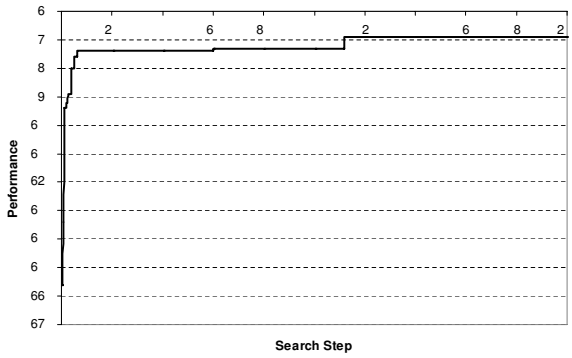

Figure 7: The highest performance index J found during each search step.

Figure 7 demonstrates that the EP algorithm is effective in searching for good policies since J is continuously improved and finally hits -569. The policy population after 200 search steps has been studied. Without elaborating on the details of every policy in the population, several high priority policies are highlighted below:

ϱ_1: The highest-priority policy for decision D1. An alternative node η is acceptable with ϱ_1 if η is relatively close to the stream source (i.e. using ψ_1 in Table 1 with e=-1.0) and has a smaller sub-tree (i.e. using ψ_2 with e=0.003). ψ_1 is in accordance with our intuition since by sending the probing message towards

the direction of the stream source, the probability of reaching a stream provider becomes higher. Similarly, ψ_2 is preferred since a node usually can quickly find its position in a smaller sub-tree.

ϱ_2: The highest-priority policy for decision D2. ϱ_2 contains three criteria, namely criteria ψ_2, ψ_3, and ψ_5 in Table 1. A node η is acceptable with ϱ_2 if it has a relatively higher capability (e=1.38) and higher available capability (e=-1). Node η should also have a larger sub-tree (e=0.39).

ϱ_3: The highest-priority policy for decision D3. Two criteria, ψ_2 and ψ_3 in Table 1, are exploited in ϱ_3. A node η is acceptable with ϱ_3 if it has a relative low capability (ψ_3 with e=1.07) and a smaller sub-tree (ψ_2 with e=0.36). ϱ_3 is desirable since positions closer to the stream source in the streaming tree are expected to be occupied by high-capability nodes. This is achieved by replacing low-capability nodes in decision D3. Meanwhile Smaller sub-trees are preferred as moving a larger sub-tree closer to the stream source may help balance the streaming tree.

Based on policy ϱ_3, the reason for an alternative node to have a larger sub-tree in ϱ_2 becomes clear. Obviously, a larger sub-tree will prevent the nodes in the sub-tree from being replaced by other sub-trees through ϱ_3. In general, it is found that the fitness of a policy is not identical to its priority. A typical example is a policy ϱ_4 for decision D1. ϱ_4 contains only one criterion (i.e. ψ_1 in Table 1 with e=-1.90), which states that an alternative node that is closer to the stream source should be selected. The fitness of ϱ_4 is the highest among all the policies in the population, because without ϱ_4 the system performance will deteriorate dramatically. Nevertheless, three other policies are to be applied for decision D1 before applying ϱ_4. The rationale behind this observation is due to the fact that ϱ_4 is only useful when node η in Figure 6 is a linker or service seeker. Despite of this result, high-fitness policies commonly will have a high priority. Figure 8 gives a partial view of the streaming tree obtained after applying the identified policy population. All stream seekers are vanished after 15 simulated seconds.

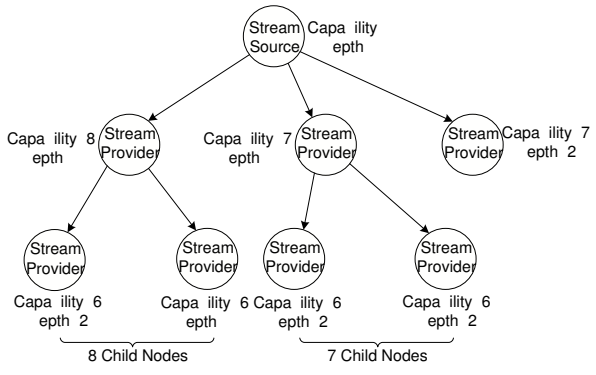

Figure 8: The streaming tree obtained through the identified policy population.

The "*depth*" adjacent to each node in Figure 8 refers to the maximum length among all the paths of the sub-tree starting from that node. Two of the three child nodes of the stream source have depth 3. The remaining one child node has depth 2. In fact, for any node η in the streaming tree, the depths of η and its siblings have a difference of at most 1. Overall, an average depth at 3.14 has been achieved. It suggests that the identified policies are effective in balancing the streaming tree. Besides, it is easy to see in Figure 8 that high-capability nodes are positioned very close to the stream source. For example, the only three stream seekers in the network that can support more than 6 child nodes finally become the child nodes of the stream source. The obtained streaming tree is nearly optimal. In fact there is less than 2% difference in J from the optimum streaming tree.

Experiments have also been performed in order to test the effectiveness of the identified policies when the application environment changes. In one such experiment, only 30% of the 300 nodes in the network are initially stream seekers. Using the policy population identified previously (denoted as $\{\varrho\}$), we found that the system performance J after 15 simulated seconds is around -204. As a comparison experiment, the EP algorithm is applied to search for good policies based on randomly initialized policy population. Figure 9 depicts the highest J found by the EP algorithm from search steps 0 to 200. The maximum J reached during the experiment is -203, which is only a tiny improvement to J obtained from $\{\varrho\}$. It suggests that $\{\varrho\}$ may lead to near-optimal performance even after changing the P2P network configurations.

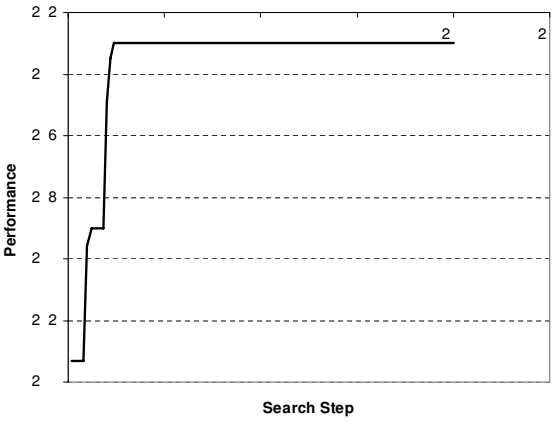

Figure 9: The highest performance index J found during each search step when 30% of the nodes in the network are initially stream seekers.

As mentioned in Subsection 3.1, policies' priority values can have a strong impact on the system performance. In order to demonstrate this impact, a simple experiment is performed with the identified policy population $\{\varrho\}$. We randomly changed the priority values of several policies in the population. The method in Figure 2 is then applied to determine the priority of each policy. Figure 10 depicts the best system performance witnessed during consecutive repetitions at step 2 and step 2b in Figure 2. As shown in

Figure 10, the performance is improved considerably after resetting the changed priority values.

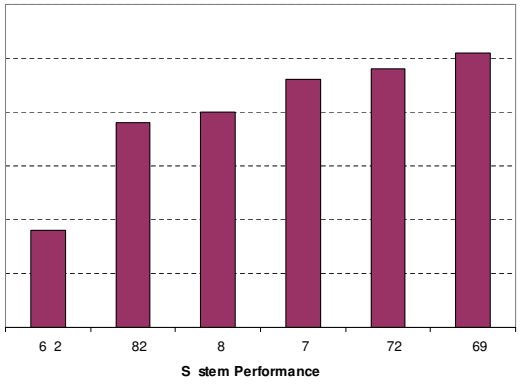

Figure 10: Performance improvement through priority evaluation.

5. CONCLUSION

In this paper, an Extremal Programming (EP) algorithm has been proposed to achieve effective search of decision policies in a open, scalable distributed system. The effectiveness of the EP algorithm was evaluated in a location-aware video streaming application. In this application, a group of nodes collaborate with each other in order to share a video stream originated from certain geographic location. A policy-based approach is utilized to guide nodes' local decisions. Simulation studies were performed and the experiment results showed that the EP algorithm was effective in finding good policies such that the average delay experienced by every node in the network was as small as possible.

Acknowledgments

This work was supported in part by the Agency for Science, Technology, and Research (A*STAR) of Singapore under SERC TSRP on IMSS Project 0521160070.

6. REFERENCES

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] S. Boettcher and A. G. Percus. Optimization with extremal dynamics. *Physics Review Letter*, 23(4):5211–5214, 2001.

[3] S. Boettcher and A. G. Percus. Extremal optimization: an evolutionary local-search algorithm. In H. K. Bhargava and N. Ye, editors, *Computational Modeling and Problem Solving in the Networked World: Interfaces in Computer Science and Operations Research*, pages 61–77. Kluwer Academic Publisher, 2003.

[4] B. Carlsson and R. Gustavsson. The rise and fall of Napster – an evolutionary approach. In *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, pages 347–354, 2001.

[5] A. J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(3):373–384, 2005.

[6] H. S. Chang, H. G. Lee, M. C. Fu, and S. I. Marcus. Evolutionary policy iteration for solving markov decision processes. *IEEE Transactions on Automatic Control*, 50(11):1804–1808, 2005.

[7] I. Cloete and J. V. Zyl. Fuzzy rule induction in a set covering framework. *IEEE Transactions on Fuzzy Systems*, 14(1):93–110, 2006.

[8] J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 768–774, 1989.

[9] J. R. Koza and J. P. Rice. Automatic programming of robots using genetic programming. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 194–201, 1992.

[10] W. Kwedlo and M. Kretowski. Discovery of decision rules from databases: an evolutionary approach. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 370–378, 1998.

[11] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(2):72–93, 2005.

[12] A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[13] P.Bak. *How Nature Works: The Science of Self-Organized Criticality*. Springer-Verlag Telos, 1999.

[14] J. R. Quinlan. Decision trees and decision making. *IEEE Transactions on System, Man, and Cybernetics*, 20:339–346, 1990.

[15] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[16] O. Ratsimor, D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha. Service discovery in agent-based pervasive computing environments. *Mobile Networks and Applications*, 9:679–692, 2004.

[17] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[19] W. T. Tsai, X. Liu, and Y. Chen. Distributed policy specification and enforcement in service-oriented business systems. In *Proceedings of the IEEE International Conference on e-Business Engineering*, pages 10–17, 2005.

[20] C. J. C. H. Watkins. Q-learning. *Machine Learning*, 8:1804–1808, 1992.

[21] B. Yang and H. Garcia-Monlina. Efficient search in peer-to-peer networks. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002.