Distributed Reinforcement Learning for Overlay Networks

SOMAYEH MASTOUR ESHGH

Degree project in
Avancerad nivå, 30 hp
Stockholm, Sweden 2011

www.kth.se
Abstract

In this thesis, we study Collaborative Reinforcement Learning (CRL) in the context of Information Retrieval in unstructured distributed systems. Collaborative reinforcement learning is an extension to reinforcement learning to support multiple agents that both share value functions and cooperate to solve tasks. Specifically, we propose and develop an algorithm for searching in peer to peer systems by using collaborative reinforcement learning.

We present a search technique that achieve higher performance than currently available techniques, but is straightforward and practical enough to be easily incorporated into existing systems. The approach is profitable because reinforcement learning methods search for good behaviors gradually during the lifetime of the learning peer. However, we must overcome the challenges due to the fundamental partial observability inherent in distributed systems which have highly dynamic nature and changes in their configuration are common practice.

Also, we undertake a performance study of the effects that some environment parameters, such as the number of peers, network traffic bandwidth, and partial behavioral knowledge from previous experience, have on the speed and reliability of learning. In the process, we show how CRL can be used to establish and maintain autonomic properties of decentralized distributed systems.

This thesis is an empirical study of collaborative reinforcement learning. However, our results contribute to the broader understanding of learning strategies and design of different search policies in distributed systems. Our experimental results confirm the performance improvement of CRL in heterogeneous overlay networks over standard techniques such as random walking.
Acknowledgments

First of all I would like to thank my advisor Jim Dowling has supervised and supported me in the right direction with his far-reaching vision. Were it not for the weekly dose of fear of disappointing him, I might never have found the motivation for the work it takes to finish. I have been inspired by him to learn more and to become a better researcher. While at SICS, I also had the privilege to interact with smart people around me.

Words do not suffice to express my thanks to my husband, Mahyar, though physically distant, were always with me when I needed him. This study is the result of your patience, encouragement and love.

Last but not least to my lovely parents who have always said I could do anything that I wanted and to my sisters from the bottom of my heart.
# Table of Contents

1. Introduction ................................................................. 4
   1.1 Problem Statement .................................................... 7
   1.2 Background ............................................................ 8
   1.2.1 From Reinforcement Learning to Collaborative RL ............... 8
   1.2.2 Distributed Learning .............................................. 12
   1.2.3 Searching in unstructured P2Ps .................................. 13
   1.3 Overview ............................................................. 15

2. Model/Implementation .................................................... 16
   2.1 Kompics Framework .................................................. 16
   2.2 Architecture of the Implementation ................................ 17
      2.2.1 Architectural concepts ........................................ 17
      2.2.2 Physical CRL Architecture, Components, and planning ........ 21
      2.2.3 Logical CRL Architecture ..................................... 22
   2.3 Summary ........................................................... 25

3. Evaluation and Results .................................................. 26
   3.1 Experiment Results with policy transformation ...................... 27
   3.2 Experiments with Advertising ........................................ 29
   3.3 Discussion ............................................................. 30

5. Concluding Remarks ..................................................... 31
   5.1 Summary ............................................................. 31
   5.2 Conclusions .......................................................... 32
   5.3 Future works ......................................................... 32

6. Bibliography ............................................................. 34
1. Introduction

The goal of this thesis is to develop and demonstrate a Collaborative Reinforcement Learning to build distributed systems that can adapt and optimize their operation to a dynamic environment for groups of cooperating, coupled agents. This area of research will advance the state of knowledge in distributed systems to achieve good performance in distributed, cooperative, multi-agent environments.

The peer-to-peer (P2P) systems have emerged by file sharing systems such as Napster and Gnutella. Due to the decentralization, these systems improved robustness and scalability, and therefore they open a new view on data integration solutions. There is however some challenges that must be overcome before the full potential of P2P systems can be realized. For example, an important feature that has to be fulfilled by p2p applications is associated with searching files, contents and resources. Currently there are two kinds of searching schemes for decentralized P2P systems: blind or informed (Xiuqi, et al., 2006). In blind search, the query is propagated without knowledge of content location, while informed search utilizes information about object locations to forward the query to nodes likely to produce answers.

P2p systems can be classified based on their architecture. The structured p2p systems, like Napster, rely on indexing all the data of all nodes in a central directory and queries use this central directory to find the desired files. In unstructured p2p systems, like Gnutella which relies on flooding are attractive for certain applications because they require no centralized directories. However, flooding is robust and reliable but highly redundant, producing many duplicated messages and much network traffic, which incurs great network load (Gatani, et al., 2005).

Several researches are currently carried out on improvements in flooding-based message routing scheme: most notable are random walk, multiple random walks and local indexes, but these strategies places most of the burden on the high degree nodes and thus potentially creates additional bottlenecks and points of failure, reducing the scalability of the search algorithm.
In this thesis, we focus on the unstructured distributed models and, to address searching inefficiencies and scalability limitations, we propose a routing algorithm that adopts a Collaborative Reinforcement Learning scheme where a collection of agents with independent action choices attempts to optimize a joint performance metric in order to dynamically change the topology of the peer network based on commonality of interests among users. Imagine, for instance, a traffic engineering application where each traffic signal may independently decide when to switch colors, and performance is measured by aggregating the throughput at all traffic stops. Problems with such factorizations where the global reward decomposes in to a sum of local rewards are common and have been studied in the RL literature.

The most straightforward and common approach to solving these problems is to apply one of the many well-studied single agent algorithms to the multi-agent problem with a very large action space. But from an information theoretic perspective such algorithms are fundamentally limited in their scalability (Bagnell, et al., 2006). In particular, the theoretical results that establish convergence and optimality guarantees for single-agent reinforcement learning algorithms do not hold for distributed (multi-agent) systems. This is because distributed environments are inherently non-stationary. Distributed systems, such as teams of vehicles, robotic swarms, or modular robots, where each individual agent has to engage in decision-making under uncertainty and coordinate their activity, present a challenge in agent design. For instance, it is possible that the agent’s policy is optimal, but the bad choices made by others prevent it from ever finding itself in a configuration from which it can move. Also, they need to be robust and adaptable to changing environments. Both these challenges can be addressed by the application of Reinforcement Learning (RL) algorithms.

However, often only the most straightforward RL techniques such as Q-learning are applied to groups of agent’s techniques that rely on the Markov property of the world. Distributed systems, on the other hand, have limitations that violate the Markov assumption: individual agents do not have access to the full state of the world, but only a partial observational window onto it (Varshavskaya, et al., 2008). This partial observability means that an agent does not have full view of the state of all other agents in the system and system performance is not directly measurable in real-time. In fact, Bernstein et al. have shown that the problem of optimizing agent behavior in such a decentralized multi-agent system has non-deterministic exponential time-complexity (Bernstein et al., 2002). Thus, most existing approaches use approximate algorithms for distributed reinforcement learning. In this
thesis we are interested in Collaborative Reinforcement Learning (CRL) in which the agents have to work together in order to optimize a shared performance measure. In particular, we investigate different feedback models, including a negative feedback model that decays an agent’s local view of its neighborhood and a collaborative feedback model that allows agents to exchange the effectiveness of actions on a sequence of joint decision.

Over the last few years there have been increasing interests in studying how to control the search processes in peer-to-peer (P2P) based information retrieval (IR) systems. In this line of research, one of the core problems that concerns researchers is to efficiently route user queries in the network to agents that are in possession of appropriate documents. In the absence of global information, the dominant strategies in addressing this problem are content-similarity based approaches. While the content similarity between queries and local nodes appears to be a creditable indicator for the number of relevant documents residing on each node, these approaches are limited by a number of factors. First of all, similarity based metrics can be intolerant since locally relevant nodes may not be connected to other relevant nodes. Second, the similarity-based approaches do not take into account the run-time characteristics of the P2P IR systems, including environmental parameters, bandwidth usage, and the historical information of the past search sessions, that provide valuable information for the query routing algorithms (Zhang, et al., 2007).

This thesis is a study in distributed systems to demonstrate that how collaborative reinforcement learning is an approach that can be used to build robust, self-managing and self-optimizing systems. We develop a reinforcement learning based IR approach for improving the performance of distributed IR search algorithms. Agents can acquire better search strategies by collecting and analyzing feedback information from previous search sessions. Particularly, agents maintain expected estimate utility on the documents for specific types of incoming queries. These estimates are updated gradually by learning from the feedback information returned from previous search sessions. Based on the updated expected utility information, the agents derive corresponding routing policies. Thereafter, these agents route the queries based on the learned policies and update the estimates on the expected utility based on the new routing policies.

The goal of the learning algorithm, even though it consumes some network bandwidth, is to shorten the routing time so that more queries are processed per time unit while at the same time finding
more relevant documents. The intention of the collaborative reinforcement learning is to adapt the agents' routing decisions to the dynamic network situations and learn from past search sessions. Specifically, the contributions of this thesis include: (1) a collaborative reinforcement learning based approach for agents to acquire satisfactory routing policies based on estimates of the potential contribution of their neighboring agents; (2) strategies to speed up the learning process. To our best knowledge, this is one of the first collaborative reinforcement learning applications in addressing distributed content sharing problems and it is indicative of some of the issues in applying reinforcement in a complex application.

The remainder of this thesis we have developed novel variations on adaptive distributed systems to adjust the system's decision-making process in order to improve its performance in future situations. The undertaking is both profitable and hard due to the nature of the platforms we study. Kompics have exceedingly large numbers of degrees of freedom, yet may be constrained in very specific ways by its structural composition.

1.1 Problem Statement

Is distributed reinforcement learning applicable in the unstructured peer to peer systems? If so, is it a trivial or a hard application? Where are the potential difficulties, and how do specific parameters of the learning affect the speed and reliability of systems? How might individuals share their local information, experience, and rewards with each other? How does sharing affect the process and the outcome of learning? What can be said about the learned distributed systems in compare with current approaches?

The goal of this thesis is to systematically explore the above questions, and provide a practical study of collaborative reinforcement learning in the p2p systems that holding massive amount of data and an efficient and scalable search for resource sharing which is a key determinant to its practical usage.

In order to understand precisely the contributions of our work, we will now describe its position and relevance within the context of the two fields of concern to us: searching in unstructured p2p systems and distributed reinforcement learning. In the rest of this introductory chapter, we give the
background on both search methods in unstructured distributed systems and collaborative reinforcement learning, describe and resolve a fundamental conflict of assumptions in those fields, present a circumscribed case study which will enable us to rigorously analysis our algorithms, and give an outline of the thesis.

1.2 Background

1.2.1 From Reinforcement Learning to Collaborative RL

Reinforcement Learning is a general method of learning problems where there is no specific correct output to train the system the right behavior (Sutton, et al., 1998). For instance, in air traffic control there may be many possible ‘correct’ answers, or the correct answers may be unavailable. These types of problems are less amenable to supervised learning. In Reinforcement Learning (RL), the system attempts to optimize it’s interaction with a dynamic environment through trial and error. Reinforcement learning provides a model to express problems of these types.

A general Reinforcement Learning model is described in Figure 1-1. At each time step t, the agent interacts with environment through the set of states S where S is the set of all possible states. According to the current state the agent would select an action ‘a’ from the available actions on state $S_t$. The Environment would feedback a numerical reward $r$ as a consequence of chosen action to the agent and the agent finds itself in a new state.
The agent progressively would learn from experience which actions should it choose. Roughly speaking the experiment is a sequence of its states, actions and rewards. To select the appropriate action the agent would implement a mapping from perceived states to probabilities of selecting each possible action. This mapping is affected by the decision-making policy used by the agent to select its actions, and thus we often end up learning something that is a function of the agent's policy. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The ultimate agent’s goal is to maximize the total amount of reward it receives in the long run.

One of the most important breakthroughs in reinforcement learning was the development of an algorithm known as Q-learning (Watkins, 1989). Q-learning algorithm works by estimating the values of state-action pairs. The value of Q(s,a) in the Equation 1 is defined to be the expected discounted sum of future rewards obtained by taking action a from state s and following an optimal policy thereafter. The Q(s,a) is estimated as follows:
Q would initialize to the fixed value according to designer estimation. Then, the agent from the current state \( s \) from \( S \), select an action \( a \) from \( A \). This will cause a receipt of an immediate reward \( r \), and changing to the new state \( s' \). Then the new Q-value would be updated the old Q-value based on the following function which calculates the Quality of a state-action combination.

\[
Q(s, a) \leftarrow Q(s, a) + x \times [r + y \times \max_{a'} Q(s', a) - Q(s, a)]
\]

(Equation 1-1)

Q-Learning usually learns with a specific rate \( 0 \leq x < 1 \) and a certain discount factor \( 0 \leq y < 1 \) through a Markov Decision Process (MDP) representation of the environment.

Collaborative Reinforcement Learning (CRL) extends RL with a coordination model that describes how agents cooperate to solve a system optimization problem composed of a set of discrete optimization problems (DOP) (Dowling, et al., 2004). It is inspired by swarm intelligence algorithms. The solution of the set of DOPs that make up the system-wide optimization problem is initiated at some starting agent or set of agents and distributed across a partially connected set of agents resulting in near-optimal use of system-wide resources. Each DOP is modeled as an absorbing Markov Decision Process (MDP).

CRL solves system-wide optimization problems by specifying how individual agents can either solve a DOP using reinforcement learning and share their results with neighbors using localized advertisement or delegate the solution of a DOP to a neighboring agent by transferring responsibility for the solution to the DOP. DOPs can be delegated multiple times across many neighbors before they are handled.

An agent delegates the solution to a DOP to a neighbor when it either cannot solve the problem locally or when the estimated cost of solving it locally is higher than the estimated cost of a neighbor solving it.
CRL Algorithm

The CRL algorithm can be used to solve system wide optimization problems that can be characterized as a multi-agent system and where agents solve discrete optimization problems modeled as MDPs in the following schema:

- A set of agents \( N = \{n_1, n_2, \ldots \} \) which are corresponding to peers a distributed system.
- There is a set of neighbor values for each agent.
- Each agent has a set of states \( s \) from the global set of \( S \).
- Each agent has a set of actions \( A_i \). In our algorithm the actions are either a delegation action or store action. The delegation action means that agent could not solve the DOP alone so it would delegate it to its neighbor. Whereas store actions are the set of DOP actions that attempt to solve the MDP locally.
- In our model, there is a cache for each agent. The value of each cache entry is a pair of \((Q(s,a), r)\) where \( r \) is the value corresponds to the last \( V \) value received from the agent’s neighbor.
- CRL model-based learning requires learning the state transition model, \( T(s'|s, a) \) that computes the probability of the action \( a \) resulting in a state transition to state \( s' \). This transition model can be provided by the user or estimated during the learning trial by observing actual state transitions. It is application dependent.

The distributed model-based Q-learning algorithm is:

\[
Q(s, a) \leftarrow R(s, a) + \sum_{s' \in S} T(s'|s, a) \cdot (D(s'|s, a) + \text{Decay}(V(s'))) + \gamma Q(s', a)
\]

(Equation 1.2)

Where \( R(s, a) \) is the MDP termination cost, \( D(s'|s, a) \) is the connection cost and \( V(s') \) is the reward value in the cache entries.

- This \( V \) value calculated by the Bellman optimality equation:

\[
V(s) = \max_a [Q(s, a)]
\]

(Equation 1.3)
Finally, a cleanup cache updater is available at each agent to remove stale elements when the cache entry drops below a specified threshold.

1.2.2 Distributed Learning

The most reinforcement learning techniques make a number of assumptions about the nature of the environment as a stationary and fully observable by the agent. These assumptions provide bounds and limitations on the learning process and its results. Unfortunately, these assumptions are usually violated by the decentralized distributed system environments which are dynamic in the real world. In any application domain involving a dynamic environment, it is not possible for the agent to have complete view of the state of its environment. Each peer only can observe a fraction of user data, and the whole data cannot be disseminated across the network due to its size. Therefore, the peers need to communicate with each other and coordinate their analysis about the data statistics they have observed. To overcome these problems we used decay and advertisement methods to keep the agent’s view updated in this partially observable environment (Dowling, et al., 2005). The following is a short description of these methods.

An Agent would advertise their V value to its neighbors whenever it executes actions and receives new V values from its environment. Each neighbor then uses the V values and their estimated connection cost to update their cache entry by new values. Different mechanisms for implementing advertisement in distributed systems include periodic broadcast of updates, conditional broadcast and event-based notification.

To maintain the agent’s view up-to-date as much as possible, we decay cache entries in the absence of new advertisement as well as after every recalculation of values. The absence of advertisements allows agent to use a cleanup updater to remove cache entries and actions with stale values in the system. The rate of decay is configurable, with higher rates are more suitable for more dynamic network topologies.

For dynamic distributed system environment, it also is important for agent to find its new neighbors and to remove old neighbors according to the environment changes. Due to the nature of distributed systems the peers may join and leave the network at any time. So the algorithm needs to be scalable.
to thousands of nodes and be robust to network churns. By discovery action in CRL it’s possible for an agent to attempt to find its new neighbors, and the decay model would remove the old neighbors as well.

### 1.2.3 Searching in unstructured P2Ps

The last few years have seen a burst of interest in the search and retrieval of distributed systems. The current peer-to-peer networks are very large; for example, the Gnutella network has scaled to more than three million nodes in 2006 (Rasti, et al., 2006) and as of May 2010, Spotify had approximately ten million users (Funia, et al., 2010). Thus, it is crucial that the algorithms are designed efficiently to scale to large number of nodes. In this section, we highlight some important types of search methods, specifically searching in unstructured peer to peer systems, because they require few constraints on topology and data placement and support more flexible search mechanisms (Denneman, 2009).

One of the first successful centralized search system was Napster which used a hybrid architecture for its information retrieval. In Napster the queries executed through a central server, while the resource demanding file transfer used peer to peer communication. However a problem would appear, when the central server shut down by various reasons, the whole system would face a single point of failure. The other centralized search techniques suffered from the same problem, they scaled poorly and could not tolerate failures. That’s why most recently developed peer to peer storage systems focused on decentralized search methods.

Decentralized searching methods can be classified as either structured or unstructured. Structured P2P networks (e.g., Chord, CAN, Pastry, and Tapestry) use directed search algorithms to assign responsibility for each file to specific peers. In contrast, in unstructured decentralized networks (e.g., Gnutella, Freenet and Spotify), there is neither a central server nor any control over the network topology and they generally use flooding or random walk also known as blind search protocols.

Structured search protocols directly route queries toward the appropriate peers for a given file through their distributed hash tables (DHT). There are a lot of advantages in directed search, such
as smaller traffic communication messages and the fact that the search fails only if there is no matching file in the system, likewise the unstructured system where it’s possible to fail the search because the algorithm could not find the appropriate peer in a reasonable number of hops. However, direct search methods suffer from limitations of DHT such as its maintenance which is impractical in some networks and DHTs only support perfect matches – if a file is spelt incorrectly then the file won’t be found.

In autumn 2008 music streaming service Spotify, which is one the flooding search techniques, gained a lot of attention from both media and public (Kreitz, et al., 2010). Spotify uses a hybrid mechanism to search for peers having content the client is interested in. The first mechanism it uses is a partial central index (Napster-style) in the Spotify back-end that maps tracks to peers who have recently reported that they have the track, and in the second mechanism peers use local knowledge (Gnutella-style) to send search requests in the overlay network. Peers send a flooding request to their neighbors and their neighbors’ neighbors.

The flooding search technique is effective in content search and allows users to perform more elaborate queries than with directed search protocols but it is very inefficient because it consumes a great amount of network bandwidth. Besides these approaches do not take into account the runtime properties of the P2P systems, such as environmental parameters, bandwidth usage, and the historical information of the past search sessions that provide valuable information (Li, et al., 2008).

To address these limitations, our routing approach adopts a Collaborative Reinforcement Learning scheme, in order to dynamically learn from the feedback information returned from previous search sessions. Peers can acquire better search strategies by collecting feedback utility from previous search sessions and from searches by their neighbors through collaborative learning. The queries would be routed based on the learned policies to the destination.
1.3 Overview

This thesis provides an algorithmic study, evaluated empirically, of using collaborative reinforcement learning by policy search with different information sources for improving search techniques in unstructured p2p networks.

At the outset, we mark out the implementation parameters and aspects of both learning and overlay networks that affect the policy search algorithm like the number of peers to be learned, the learning parameters and the bandwidth of the network. Most of the content presented in this thesis is related to at least one of these issues of influence in order to increase the speed of finding queries in unstructured overlays and the quality of the learning algorithm output.

The basic idea behind our novel proposed algorithm is to reduce the number of peers that process a query to as low as possible. Our experimental results will show that practical queries can be answered by fewer peers than the current techniques. For example, we compare our results with the random walk technique which is one of the most common blind searches in distributed systems.

We quickly find that learning in distributed systems where the agent has partial observability of the system require careful construction but ultimately our simple suggestions such as advertisement and decay model compensate these limitations and can provide good starting points to policy search. This solution is thus a systematic way of generating good starting points for distributed learning.

Finally, we present a new algorithm with which individual agents in distributed systems can finally reach agreement on both rewards and experience gathered during the learning process which will result in consistently better policies and faster learning. We develop our implementation in a novel framework named Kompics that simplified the development of our complex distributed application. And we evaluate our claims and algorithms on a number of various scenarios for different search policies to illustrate Crl search algorithm can give close to optimal performance with much lower time complexity than exhaustive searching in distributed systems.
2. Model/Implementation

This chapter presents the detailed implementation model that we have used for our CRL algorithm. Section 2.1 introduces the Kompics framework that our model builds upon (Arad, et al., 2008). It is important to have a basic understanding of this new framework in order to understand the implementation. The other sections are dedicated to describing our prototype’s design and its functionality. We think that the readers of this document have sufficient knowledge about the reinforcement learning aspects so we will not discuss CRL concepts at length in this chapter however the main concepts which are used in the implementation would be covered.

2.1 Kompics Framework

The Reactive Component Model for Distributed Computing (Kompics) framework was started in 2009 in Sweden at the Royal Institute of Technology (KTH) and the Swedish Institute of Computer Science (SICS). Its goal was to simplify the implementation of distributed systems which are complex due to concurrency and partial failures. This novel framework simplifies implementing distributed protocols by splitting them into multiple reactive software components. The components in Kompics are state machines which interact with each other through ports. These ports are for passing typed events between components connected by channels. In this section, we will briefly introduce the conceptual entities of the Kompics framework.

- **Components**: A component is a state machine that contains some internal states and a set of event handlers which are specific procedure to execute the received events. Components interact with each other by triggering (sending) and handling (receiving) events.

- **Events**: Components in Kompics can "subscribe" to receive certain specific events from the channels. As an event is "published" to the channel, it will be propagated to all appropriate subscriber components. This allows components to be loosely coupled.

- **Channels**: The interaction between components occur through interaction links called channels in Kompics. They are responsible to send published events to appropriate components.
• **Event handlers**: All event handlers are software procedures that are invoked on its associated component in response to a specified event being received. Components subscribe their event handlers to channels by registering event subscriptions at the respective channels.

• **Component fault isolation**: All unhandled exceptions that are not caught in the event handler are caught by the runtime and wrapped into a fault event. This fault event will be published into the component’s control channel. Then a supervisor component can manage this faulty component.

### 2.2 Architecture of the Implementation

In this section we present the architectural pattern of our CRL design. The implementation is based on component patterns of Kompics. We’re first going to introduce some basic concepts and then we’re going to start expanding some of our design components.

#### 2.2.1 Architectural concepts

**Discrete Optimization Problem (DOP)**: According to heuristic search algorithms a DOP is a tuple \((S,f)\) where \(S\) is a set of possible solutions and \(f\) is the cost function that computes the cost of each solution as a Real number. The objective of DOP is to find a best solution path (with minimum cost) in a graph from an initial node to a goal node. We simulated DOP in our simulation as a message that transfers set of feasible solutions and costs from one peer to another peer through the channels.

**Markov Decision Processes (MDP)**: MDPs are characterized by mappings for a set of states, actions, Markovian transition probabilities, and rewards. The goal of Markov decision process is to find an optimal policy (figure 2-1). We will describe the set of states, actions and policies that can be used to make optimal decisions in our CRL model implementation.
• **Action:** In our implementation we consider three types of actions: Delegate, Discover and Store Action. Whenever an agent receives a DOP, it has to decide what action should be used to solve this DOP according to its action selection policy. This decision is very important because the search queries success depends largely upon a number of good chosen actions of the agents. In CRL if the agent peer couldn’t solve the DOP directly then it would delegate the DOP to one of its neighbor (Delegate Action). Otherwise the Store Action would have chosen if the agent can adopt the DOP by itself. The Discovery Action is used for finding dynamic new neighbors in distributed overlays.

• **State:** When an agent chooses a new action it would receive a new state from its environment. The action selection would cause a state transition to one of these new states: Delegated, StoredLocally or TTLExpired. It’s easy to guess that Delegated and StoredLocally states are related to the description of their mentioned above actions. TTLExpired state is when the DOP could not solve through the specified threshold hops in the network.
• **Reward**: The ultimate goal of a learning agent is to collect more reward through executing good actions. Rewards are real numbers that are received from the environment as a feedback to approximate the probability of correctness of agents' decisions. Intuitively, a good reward function is one that gives the agent useful feedback. Our reward function would return back the highest reward if the agent solves the Dop locally which is the most important accomplishment for the search policy. However the function returns negative reward as punishments if the Dop reached the TTLExpired state which means that the Dop was not solved in the appropriate expected time.

**Policy**: In order to maximize the sum of rewards over states an agent should select good actions through a learning policy. We used the most popular policies namely greedy, Boltzmann and random search (Sutton, et al., 1998). We describe below the key features of these policies. In the analysis chapter we’ll focus on these policies and explain where and how applications can use these policies.

• **Greedy Policy**: Consider quadruple \( (s, a, r, s') \) where action \( a \) is taken in state \( s \), resulting in reward \( r \) and a transition to new state \( s' \). And remember from the first chapter that Qvalues are updated in a tabular format in the cache according to Equation 1. The greedy policy \( \pi_g(s) = \text{argmax}_a Q(s, a) \) is optimal when the Q values are accurate.

• **Boltzmann Policy**: One of the important policy method used in reinforcement learning relies on a Boltzmann distribution as defined in Equation 2-1. Control on exploration could be adjusted by a temperature parameter \( \tau > 0 \). For example at very low temperatures Boltzmann distribution is equivalent to greedy policy. Boltzmann doesn’t always pick the 'best' action like the greedy policy. It chooses better actions with higher probability.

\[
\pi(s, a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}
\]

(Equation 2-1)
In the next section we study deeper the CRL model components and policies learned with agents in the unstructured distributed overlay. The description can be divided into two parts: how to build a distributed overlay that minimizes the query response time, and how to use CRL techniques in order to perform a search in the best way.

**Cache:** Each peer has a unique cache object that stores data so that future requests for that data can be served faster. In reinforcement learning design our cache model is referred to as QModel interface. Each cache entry stores State, Action and Q-values associated with delegation actions. The Q-values correspond to V-values advertised by other Crl agents. The Q-values are decayed over time and removed from the cache when they exceed a deletion threshold.
2.2.2 Physical CRL Architecture, Components, and planning

Since Kompics is a component based platform, we’ll describe the implementation by components according to figure 2-2.
The **CRL Experiment** component in our Java implementation is actually the Java main-class. When first executed, it will invoke the Kompics runtime system. Executing the CRL experiment and launches CRLSimulationMain, that, in turn, is responsible for creating bootstrap server, CrlMonitorServer and Peers and their configuration methods. CRLSimulation also initializes the CrlOverlay configuration to set the parameters such as which policy should be used, dop timeout, cache period and advertisement period.

**CRLSimulator**: This component creates and starts the Crl peers upon to the receiving of the Join event. Join event publishes to this component whenever the new peer wants to join our unstructured distributed overlay. CRLSimulator component gives a unique IP address to each peer and a list of previously joining peers. This list is used to construct our neighbors view in the CRL Agent component. Our data statistics such as the percent of successful search queries, average number of hops and number of failed queries are also handled in this component.

**CrlPeer**: Each peer creates CrlAgent class and its own BootstrapClient, CrlMonitorClient and CrlWebApplication.

**CRLAgent**: Most of the implementation part is the each agent class. At first the agent tries to construct its view which is list of its neighbors. As we said before the CRLSimulator would pass a linked list of all available peers to new agent. The overlay we created for our experiments is a matrix, so the agent chooses appropriate peers from the list so that it can construct the matrix overlay.

### 2.2.3 Logical CRL Architecture

Here we’re going to cover our CRL algorithm. The algorithm starts to construct the matrix overlay as mentioned before according to the algorithm 2.3.1. In this algorithm, each agent chooses its adjacent peers in the matrix as a neighbor.
Algorithm 2.3.1 Agent’s View

\begin{algorithm}
\For{all insider $\in$ linkedlist}{
    peerNum $\leftarrow$ getNodeNum(insider)
    selfNum $\leftarrow$ getNodeNum(self)
    \If{peerNum is on the top, right, left or down of selfNum}{
        addneighbor(insider)
    }
}\end{algorithm}

The main part of the algorithm is related to processing the Dops. According to Algorithm 2.3.2, the first peer that discovers the Dop, looks the problem id to see if it has it locally or not. If requested data is contained in the data list, this request can be solved by simply updating the cache values by the best reward obtained from the environment and send a report message to the Experiment component to show that the Dop is solved. To approach optimal learning the agent sent a feedback message to visited peers list stored on the Dop. However, if the agent doesn’t have the requested data it will add itself to the list of visited agent in the Dop. Then the agent tries to contact a peer in its neighbor view chosen by the learning policy calculation. To prevent the same message from being continuously rerouted to the same peers over the network, the agent checks the chosen peer with visited peers list on the Dop. If the found peer already exist all the calculation ignored and the agent executes the learning policy again to choose a new neighbor.

When a contacted peer receives a message, it performs the following steps:

1. The peer increments the number-of-hops field on the Dop.
2. The peer checks the number of hops allowed. If the Dop exceeds the maximum number of hops, the dop is discarded. This helps to prevent the CRL to go through thousand undesirable peers for invalid queries.
3. The peer sends its vValue from the cache and an acknowledgement message to the correspondent agent.
4. It proceeds the Dop according to the Algorithm 2.3.2.
5. The Dop goes back and forth through these steps until a peer satisfies the request.
On the other hand, upon receipt of the mentioned acknowledgment message in step 3, the recipient agent updates its cache with the new vValue. The cache is updated with the new vValue just if this new one is better than the current vValue.

---

Algorithm 2.3.2 Agent Dop process

1. **Initialize** parameters according to experimental condition
2. **for each** DOP **do**
   1. **if** DOP exist locally **then**
      1. **if** Dop visited other peers **then**
         1. get global best reward R
         2. update the cache by the new R
         3. get vValue from the cache
         4. Av = pop the previous visited agent from the list
         5. **trigger** vValue as a feedback to Av
         6. **trigger** new message to CRL experiment _Dop is solved successfully
      2. **else**
         1. **trigger** new message to CRL experiment _Dop is solved immediately
   3. **else**
      1. bestN = Calculate policy () to choose the best probable neighbor
      2. add itself to the visited peer list stored on the dop
      3. delegate the dop to the bestN
4. **end if**
5. **end for**

So far the cache, which is actually a partially observable model of our environment, is updated under two circumstances: Either by the immediate acknowledgments from the delegated neighbors, or by the final agent who answered the problem with a feedback value according to the algorithm 2.3.2. To improve the performance of the routing policy, we used decay and advertizing methods to keep the agent’s view updated as fresh as possible (section 1.2.2).
Each peer sends periodically its V-value to its neighbors. The neighbors perform cache update function upon arrival of these messages. For the decay mechanism, each agent has a local timer that periodically calls cleanup updater to remove cache entries and actions with stale values in the system.

2.3 Summary

In our CRL model, we used Kompics framework to separating the code into several components to have a clear implementation of each part, and to have the ability to change one part without altering the rest of the code. We have introduced this loosely based framework in the first section of this chapter.

We have formulated different parts of our protocol as a multi-agent solution for gaining the best search by RL policies. We discussed these issues in the next chapter by introducing key features of the CRL implementation such as DOP, MDP, and Cache. We have identified our component-based architecture design and explored the information and constraints that can affect the CRL performance in the CRL algorithm description.

In the next chapter, we explore our experimental results and evaluations to validate our claims. Finally, we discuss the significance results of this approach in chapter 5 and conclude with directions for future work.
3. Evaluation and Results

What is the effect of different policies on a CRL overlay? How feedback rewards and advertisement can improve the search result? This section analyses the analytical performance that answers these questions of the distributed architecture introduced in the implementation section. The evaluation issues related to the study of CRL can be divided into two categories: The first one analyzes policy transformation and the second analyzes advertisement effects. Through this analysis we find: 1) Determining the ideal search speedup obtainable using best policy, 2) Explaining the effectiveness of advertisement on the number of succeeded queries from which we prove that its benefits overcomes its overhead message passing.

For our simulation, we have used Kompics, a novel simulator for distributed systems introduced in previous chapters. It can simulate with Java source code and Apache Maven to organize and manage its artifacts. One of the important features of this simulator is that it provides a special scheduler for system simulation which means that the system code is executed in deterministic simulation provided it does not create threads (Cosmin, 2010).

The following notations are used in this chapter:

- $N$, number of peers in the overlay
- $View$, the table of neighbors on each peer
- $Q$, the Q-value
- $TTL$, number of maximum hops
- $t_{adv}$, Advertisement time scheduler
- $t_d$, Decay time scheduler

In this experimentation, we set up a network of 900 peers with thousands of unsolved DOP problems. Each peer joins the system in every second according to our simulator scheduler timing. After the creation of the overlay, a sample song would be added as a data to one of the randomly chosen peers. As mentioned before peers are in the neighboring view of their direct peers in the 30*30 matrix overlay.

The Dops packets are sent by the simulator every 100 millisecond follow an exponential distribution. Each Dop has its own problem id, these problem ids actually can be defined as a user query that looks for the special data. The initial Q is -100 and TTL is set to 40. Finally, the decay rate $t_d$ and $t_{adv}$ which discussed in previous chapter are set to 10 seconds.
3.1 Experiment Results with policy transformation

So far we have looked at the potential effects policies might have, but here we're going to analysis the substantial effects of these policies on the final results of routing quality. In our tests, we have used three different policies; Greedy, Boltzmann and Random.

Because of the many possible ways of looking at the data, we just focused on the average number of failed queries which is one of the main elements, at least in our opinion, for discussion here. Figure 3-1 illustrates this comparison.

As figure 3-1 shows, there are significant differences in different policies when number of queries are increased. In this figure each color represents a different policy and each bin consists of hundred diverse Dops. We let the application run for ten bins equal to 1000 Dops, measured the difference of learning progress, and it appeared that after this duration the learning converged to fix number of unsolved queries. That's why we decided to keep the duration of our experiments up to thousands Dops.
The Random walk policy has the most number of unsolved queries according to the figure 3-1. In random walk interpretation, we are less likely to find a route to the destination peer without learnability, while the learning provides a natural way of exploiting this knowledge.

CRL is based on observing the environment changes to find the optimal path given an initial state and a goal. The major difference between Greedy and Boltzmann policies are exploration and exploitations concepts in Q-learning. For the greedy policy, the action selection is based purely on current values of the state-action pairs (Exploitation). In contrast, Boltzmann strategy is the strategy based on the assumption that the agent selects a nonoptimal action in the current situation and obtains more knowledge about the problem (Exploration). This policy allows the agent to reach the globally optimal by neglecting the local optimal policies instead (Guo, et al., 2004).

In the next section, Figure 3-3, we’ll see that without advertisement the Greedy policy is identical to random policy. In our CRL algorithm, when we just used exploitations the performance of learning algorithm decreases with respect to the results. The problem is that the greedy policy always chooses the action that has the highest estimated value. At the very beginning, this method improved slightly faster but then leveled off comparing with Boltzmann policy, because it often gets stuck exploring the new value of other neighbors. In contrast, models that are based on Boltzmann or similar exploration processes are more appropriate in our CRL model and thus having in effect a more restricted learning ability compared to both the random and the greedy learning models. This is also visible in Figure 3-1.
3.2 Experiments with Advertising

Figure 3-2 Boltzmann policy

Figure 3-3 Greedy policy
As pointed out before, we introduce new advertising method to handle the multi-agent partially observable environment. This method updates the state/action pairs periodically to obtain a high quality learned behavior between different policies defined.

In the previous section, we achieved high performance by using Boltzmann in compare to other policies. This could be an indication of either strength in the learning policies or in the superior advertising quality, but we cannot state the most contributing factor without further testing. Here we compare the performance of our approach with and without advertising against the Greedy and Boltzmann policy. We notice that our approach has the lowest unsolved queries than the CRL policies without advertizing and the reduction of the failed Dops to 30 percent of the original generally causes a significantly higher level of routing satisfaction than the pure learning policies. Figure 3-2 and 3-3 show the effects of advertising between the agents confirming our impression that advertising method is at high level of importance to the perception of quality.

3.3 Discussion

We have demonstrated the applicability of CRL to unstructured distributed systems, and presented the effect of different learning policies on these overlays. As we move from single agent to multi agent environment, peer only can observe a fraction of user data, we quickly outgrow a model for their coordination. Therefore, in this chapter we have introduced our advertizing model that allows agents to communicate and reduces the number of unsolved queries.

With the evaluation of our results on different policies functions, we have also presented a study of the behavioral knowledge of different learning policies. We find that the Boltzmann policy has the best performance in our CRL algorithm comparing with two other policies. Finally, we demonstrate empirically that advertizing method can provide even higher performance which is a good starting point for further policy routing.
5. Concluding Remarks

5.1 Summary

In this thesis, we have described the problem of information retrieval in distributed systems through collaborative reinforcement learning.

We have demonstrated that applying learning techniques to distributed systems is technically difficult where each agent has only access to its own local observations and a local estimate of reward from environment. We then proposed a class of algorithms to mitigate these problems through learning and to obtain the higher searching performance in compare with currently available solutions. We have shown that our advertizing solution is more capable in situations where the learning policies like random and greedy are not, because it has the strong assumption of full observability.

We identified the key concepts using in information retrieval for learning, and specified the important parameters and their effects on the distributed overlays. We then discussed a novel component base framework for distributed systems. We explained our approach and then described our implementation from both the logical and physical architectures.

Along the way we provided experimental evidence for our algorithms' performance on the policy transformation, and further evaluated our results to address the strengths of our advertizing algorithm, which we have shown speed up learning comparing with previous ones.
5.2 Conclusions

In this thesis we learned that using the CRL in unstructured distributed overlays with the partially observable environment is extremely complicated, but it is possible to use policy search and advertising method to learn reasonable behaviors.

From the test results, we have learned that the learning policy influence the performance of search queries. As with choosing the better policy, the walk steps to find the target agent in the overlay decreases, causing faster response time to find the content. Also we have theorized about the significance of having advertizing with a learning policy to achieve higher possibility of finding the content of search query successfully.

The Information retrieval performance affected by lot of parameters, in particular, by the quality of the reward estimate policy. It is possible to configure the best reward values in experience results of searching performance by good design strategies such as using appropriate policy representation. The good design for mitigating the difficulties of partially observable systems by CRL requires careful human participation and trade-offs between number of parameters to learn, the initial configuration of the overlay, and the search constraints.

While this thesis is mainly concerned with learning policies on a matrix overlay, the results obtained herein are in no way restricted to this scenario. This overlay generalization required only minimal additional assumptions to ensure that good policies would be result in the same good performance results of information retrieval. That means the structure of the algorithm is the same, and the lessons we have learned about CRL remain valid.

5.3 Future works

As we explore a simple performance measure such as reward function can always improve the efficiency of the search algorithm. In this thesis, we have restricted experiments to CRL learning in order to compare the different approaches to learning functions. Future work could include more sophisticated reward algorithms.

We have also seen throughout that the learning success rate depends not only on network size, but also on the initial configuration that we assume at the start of every episode. In this thesis, we have limited our study to a few such initial configurations to focus on the effectiveness of learning instead. In the future, we would like to describe the space of initial conditions, and research their influence on learning performance more broadly.
6. Bibliography


Bagnell J. Andrew and Ng Andrew On Local Rewards and Scaling Distributed Reinforcement Learning [Journal] // ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS. - 2006. - pp. 91-98.


