Find a Stochastic Path in a Robust-First Distributed System

Xinyu Chen

University of New Mexico, Albuquerque, Albuquerque, New Mexico 87131
xychen@cs.unm.edu

Abstract

To find a robust-first communication approach in distributed systems, we present a case study of long distance path-finding in the Movable Feast Machine(MFM). The model let computation requests follow some signals to find the service providers. Instead of using location-dependent addresses or relative coordinates, These signals can indicate the gradient of themselves so that they form stochastic paths towards the source. Such paths are self-constructed and self-repairable. This case study helps to extend communication distances outside Event Windows. This is a step towards a more robust distributed system.

Introduction

determinism is always preferable because it seems simpler and more efficient than uncertainty. Traditional computer architectures based on this principle have helped to deal with many real-world problems. But the simplicity and the efficiency here are not assured because the real world is full of uncertainties. Such uncertainty comes from the variety and redundancy of this world. Not a single event can destroy the ecosystem. In a sense, the uncertainty represents a robustness. This feature of the ecosystem is appreciated by computational systems. It is time that we begin to think in a robust-first way to make life-like systems. Then we can truly accomplish scalable and high-performance computations.

Our model is designed on the framework of the Movable Feast Machine (MFM). The Movable Feast Machine is an indefinite scalable distributed system with the robust- first thoughts. To achieve the indefinite scalability, the MFM is designed to distribute processors and memories in spatial grids while still processing regular structures such as short spatial dependencies and error tolerance (Ackley et al., 2012). The computation tasks are fulfilled by asynchronous Cellular Automata, also called *Atoms*, which are instances of *Elements*. A Element in MFM is like a class in an object-oriented language. It describes the behaviors and other properties Automata. When given an event, an Atom can operate a limited sites called the *Event Window* in a parallel manner. The Event Window defines both timporal and spatial restrictions. It restricts Atoms to interact with a limited number of objects. Thus every objects in MFM are equal and they are only responsible for their own behaviors. There is no super units that controls global status. This lifelike nature of the MFM gives the computations on it the ability of self-organize and self-repair and eventually the robustness. However, to achieve such robustness in MFM is not easy. We have to change our serial thinkings to spatial thinkings and random thinkings. The behaviors of Aotomata must be reasonable with limited knowledge of the environment and limited time constraints.

Path-finding is a useful subject both in real- world applications and theoretical researches. These include transportation, communication, game design, computer networks and so on. However, with the light of robust-first thinking, we need to reconsider this problem under spatial distributed conditions. The uncertainty of the events order and the asynchronous feature in distributed systems give many path-finding algorithms great challenges. Without global addresses, how can algorithms construct paths? On the other hand, with global addresses, can these systems really be scalable? Some primitive methods that we try to avoid under efficient-first context need reevaluations under robust-first concerns. Such methods include and not limited to bubble sort and broadcasting communications. With no surprise, some simple mechanisms work robustly in the biology environment.

Inspired by biology, we find such effective examples of path-finding. An firefly sends out flash signals to attract partners, an E-Coli takes biased random walks towards food source, neuronal growth cones are guided by both chemoattractants and chemorepellents *in vitro*. All these reminds us some path-finding mechanisms by the gradient of signals or medium. In this case study, we use a signal-emit and gradient-check path-finding approach like broadcasting communications. To achieve some generality, the model has three abstract participants: *Requests*, *Signals* and *Providers*.

Requests diffuse in the given space of MFM until they find a Provider and disappear. We consider in this situation a Request is satisfied. Signals also diffuse in the universe of the MFM. As pheromones, they have an evaporate rate.

This feature gives the whole collection of Signals a cloud-like shape. The density of Signals around the Providers is higher than the Signal density on the edge of this cloud. This difference of density can be seen as the gradient towards the Providers. Our intuitive idea is to let Requests detect and follow this gradient towards the location with the highest density. Hopefully, this location is close to Providers so that Requests can find them. To detect Signal densities, Requests count and record the number of Signals they see in the current Event Window. By the same method, Signals also detect and report the number of their fellow Signals in their Event Windows. By comparing this densities, Requests can decide the approximate direction of Providers. Due to the asynchronous feature of the MFM, this comparison is between the density observed by Requests in the current event with the density records reported by Signals some events before. However, this detect-and-compare approach still improves the success of Requests' path-finding.

The success of path-finding can be measured by the percentage of Requests that are satisfied within a given distance between Requests and Providers and a given period. The distance restriction is a prerequisite because Signals will evaporate. They can not cover a big area before they die. On the other hand, if they live too long they will sparsely exist in some Requests' Event Window. In both cases, they can not indicate the gradient well. Thus it is reasonable to limit the distance between Signals and Providers. The restriction on a given period is also necessary because Requests are taking random walks towards Providers. With less and less Requests left unsatisfied, the probability for one of them to find Providers drops. It takes much longer time for the last Request to be satisfied than average. So we add this time constraint in order to show a trend of the effectiveness. By adjusting the evaporate rate and emission odds of Signals, we obtain different Signal strength, which is represented by the pupolation of Signals in the universe of the MFM. Thus we can measure the the effectiveness of path-finding under various Signal strength.

Model Descriptions

Our path-finding model relies on the interactions between Requests, Signals and Providers. To simplify the situation, only Requests can move and they search for Providers. We set the Providers in a fixed area. However in real-world applications, requests and providers are dynamic. We can see this mobility from the usage of smart phones and other mobile devices. Requests and providers may continuously vanish and appear. By the contrast of the moving Requests and the static Providers, it looks like that Requests are captured by Providers. In fact, Requests are positive in this interaction. In the MFM, we use Element behaviors to implement the interactions described above.

Provider Behaviors

Providers are responsible to send out Signals. They have only one primary behavior. This *SignalEmit* behavior decide how often Signals can be emitted and also the evaporate rate of Signals. We can adjust these two parameters to get various density in the universe of MFM. The *EmitOdds* is set from 0 to 10. The chance of emission is $EmitOdds \times 10\%$. In this context, Providers send out no Signal in their event when the *EmitOdds* is zero. They create a Signal in each event when the *EmitOdds* is 10. Although the evaporate rate is set by Providers, the actual behavior of evaporation will be described next.

Signal Behaviors

Signals are the medium of this stochastic path. They have three behaviors :*CheckDensity*, *Evaporate* and *Move*. When a Signal Atom get an Event, it first checks how many other Signals exist in the Event Window. Further behaviors depend on the observations of fellow Signals. We set the initial density to two when the Event Window contains only one Signal Atom. The observation of density is

$$0.1 \times NumProvider + NumSignal + 2 \qquad (1)$$

We set this number start from two based on the following consideration. The Signal Atom itself should at least count for density one when there is no other Signal in its Event Window. When a Request meet a single Signal, the Request's observation is also density one. To let the Request move towards this Signal, we raise the initial density. This situation happens a lot at the edge of cloud and the initial value of two has no other side effect. Also we reward Signals near Providers a bit by count Providers as Signals too.

Although the *EvaporateRate* is set by Providers, this *EvaporateRate* is only a base number. The actual probability of *evaporate* is determined by the product of this base number and the observed density. The Signals that are far from Providers should evaporate faster than Signals that are close to Providers. So the probability of evaporate is

$$PrEvp = \begin{cases} \frac{1}{|EvRate - Obsv|}, if obsv \leq 2 \\ \frac{1}{|3 \times EvRate - Obsv|}, otherwise \end{cases} \qquad (2)$$

Decided by the probability of evaporation, Signals destroy themselves before moving further.

The last behavior of Signals is *Move*. If a Signal doesn't destroy itself, it decides how to move. At this moment, it has a record of density to report to Requests. If it meets no Request in the current Event, it diffuses. Otherwise it stays and reports the density it observed. We design this stay behavior for Signals because we want the observation of Requests and the observation of Signals to be as close as possible. If we let Signals diffuse at any time, Requests can only read the history observation of Signals. However, due

to the asychronous feature in the MFM, this design is only an attempt to keep observations from both side close. We can adjust this in the future.

Request Behaviors

Requests are quite similar to Signals. They have also three primary behaviors : *Evaporate*, *CheckDensity* and *Move*. For Requests, the most important thing is to find Providers and get satisfied. As soon as a Request is *captured* and satisfied, it disappears. In this sense, Rquests can evaporate too. This evaporation is not determined by some odds but by the presence of Providers. We design the *Evaporate* happens at the distance of one. This restriction is based on the assumption that too much Signals can hinder the *Evaporate* process. In the future works, we can remove this limit and let Requests find Providers in their entire Event Window.

Like Signals, the Requests' *CheckDensity* behavior is also a precondition for their Move behavior. When a Requestis given an event, it counts the number of Signals in its Event Window. If there exists some Signal Atoms, the Request can read their density reports. The Requests always has the accurate density in its current Event Window. While the reports from Signals may come from several events before.

The *Move* Requests contains three subbehaviors: *Swap*, *TryMemory* and *Diffuse*. The *Swap* happens when a Request finds a Signal in their Event Window reports a higher density than its own observation. In this case, the Request swaps with that Signal. At the mean time, it remembers the offset of this Signal. The memory of Requests' recent swap offset helps Requests to take *TryMemory* behavior. If a Request finds no Signal in the Event Window, it has some probability to move according to its memory of recent swap. This is the *TryMemory* behavior. Hopefully they can find more Signals along the memory direction. The probability is set to 0.5 because We discover some bad luck Requests fly out of the universe with their memory if they never meet any Signals again. They have a 0.5 probability to erase this memory and *diffuse*.

In the following experiments, we first set the *EmitOdds* zeor as our baseline to compare the effectiveness of our path-finding process.

Experiments and Results

Acknowledgements

I want to thank Professor Ackley at the Department of Computer Science at the University of New Mexico. And also my classmates in the Artificial Life lectures in the Fall 2014. Without their help, even a simple case study like this is not possible.