P2P-Based Scalable Execution Platform for Algorithmically Transitive Network

Mikio Yoshida[1], Hideaki Suzuki[2], and Hidefumi Sawai[2]

[1] BBR Inc.
2-1-4-206, Sonezakishinchi, Kita-ku, Osaka, 530-0002, Japan
yos@bbr.jp
[2] National Institute of Information and Communications Technology
588-2, Iwaoka, Iwaoka-cho, Nishi-ku, Kobe, 651-2492, Japan
{hsuzuki,sawai}@nict.go.jp

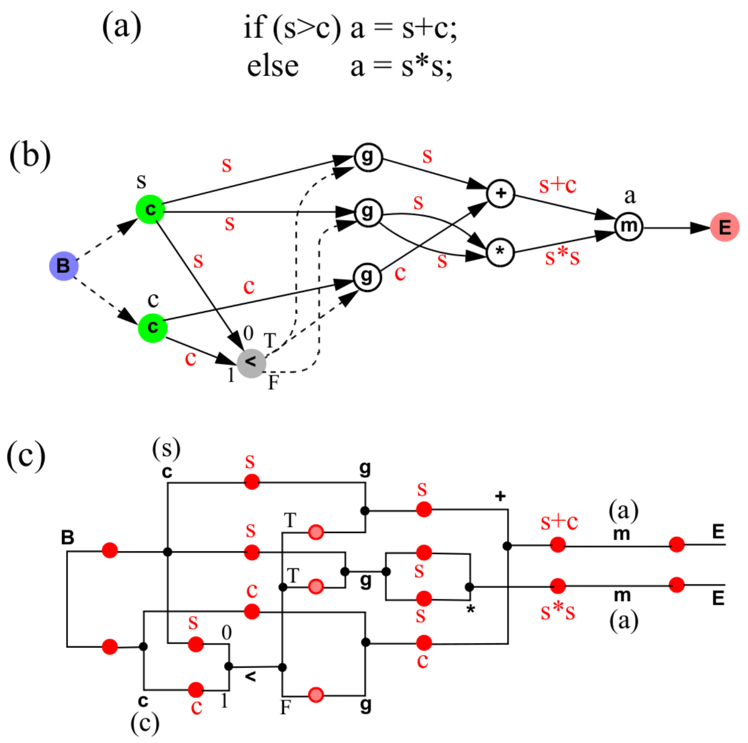
Abstract. "Algorithmically Transitive Network" (ATN) is a novel computational model based on a data-flow network, consisting of the following operations: a forward propagation propelled with node firing and token creation, a backward propagation caused by evaluating differential coefficients, and a topological alteration taken place by autonomous agents. In the research of the ATN, a simulation run on some parallel processing scheme is essential. As a flexible and powerful implementation scheme, the paper employs a P2P based distributed platform, and describes the mechanisms for simulation and P2P deployment of the ATN. The implemented platform has the following three features: flexible allocation of ATN nodes to the physical resources, unified description of communication between nodes, and several methods to realize high parallelism. The proposed scheme is also helpful to verify applicability of the employed P2P system.

Keywords: peer-to-peer network, framework system, data-flow architecture, agents, oneway RPC.

1 Introduction

"Algorithmically Transitive Network" (ATN) [1] [2] is a novel computational model based on the data-flow network [3]. The ATN consists of operations of the bi-directional propagation of 'tokens', a forward propagation (FP) which advances calculation ahead using 'firings' of nodes, and a backward propagation (BP) which evaluates calculation to the opposite direction of the flow of tokens in the FP.

As in the data-flow network, the ATN's nodes read the input tokens on their incoming edges, fire, and create the output tokens on their outgoing edges during calculation. This constructs a 'fire-token pedigree' whose nodes (tokens) represent variables or mathematical expressions, and whose hyper-edges (firings) represent arithmetic/logical operations used to create the tokens. An example of these relationships is shown in Fig. 1. The BP, which is propelled on the pedigree, finally modifies parameters of the ATN at some nodes, whose statistical data is used to change the network topology by functional agents distributed in the network. The aim of the ATN research is in making the data-flow network itself learn through these operations and explore novel algorithms automatically.



```
(a)     if (s>c) a = s+c;
        else     a = s*s;
```

Fig. 1. (a) Higher language program of a simple conditional branch, (b) data-flow network (ATN), and (c) fire-token pedigree produced by the calculation. The variable name `s` represents the graph's input (sensor) signal, and the `a` represents the graph's output (answer) value. The pedigree's top ancestor is the initial fire at the '`B`' node, and its last descendants are firings at the '`E`' node or nodes with no outgoing edge. In (b), arithmetic edges are expressed as the solid arrows, and regulating ones are expressed as the broken arrows.

Of course, an implementation and an experiment of the ATN can be conducted on a single core computer; however, in order to precisely evaluate the ATN's ability to explore algorithms, implementation to parallel computers, such as a PC cluster with sufficient calculation resources, is essential. Although now in many cases, MPI [4] is used as a programming tool for a parallel computer, here we propose building the ATN on a P2P platform system named "PIAX" [5]. There are three advantages for taking this approach. First, a PIAX's higher level library provides an efficient programming environment for software developers as well as the software's flexibility. Second, the ATN which uses the data-flow network for the base of calculation has high affinity with a P2P network, requiring the minimal communication cost between CPU cores. This will be also enhanced by the trait of the PIAX able to use threads efficiently and utilize the core resources for the maximum. Third, the P2P library enables the ATN to be deployed to the open network environment in the world.

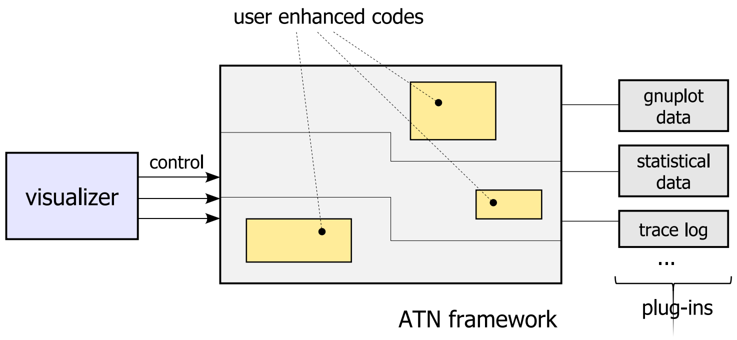


Fig. 2. Modular structure of ATN-P2P

In the research of the ATN, what kind of agents should be prepared and what kind of topology changing rules should be made are important research agenda. To inquire about these issues, we have to try many agent designs with repeating each evaluation and verification. Moreover, to confirm the ATN's high parallelism and functionality as a distributed system, it is also necessary to check about the performance of the ATN implemented onto an actual P2P network. The P2P-based platform system we have developed ("ATN-P2P") not only supports the research of the ATN from these points of view but also clarifies the outcome of or difficulty in deploying the ATN to a P2P network. Subsequent chapters describe the design and implementation of ATN-P2P, focusing on mechanisms for the ATN simulation platform and P2P deployment.

2 Basic Design of ATN-P2P

2.1 Framework and Modularity

The ATN-P2P has a form of a 'framework system' so that an ATN researcher can try and implement an idea freely. The researcher can change, add and delete the following functions without caring about the system behavior of the topology change of the ATN and the detailed flows of tokens in the FP and the BP.

– Insertion and deletion of a node, and the change of an operation defined in a node.
– Change of calculation of differential coefficients and teaching signals.
– Insertion and deletion of an agent, and the change in behavior of an agent.

In addition to implementing a new function, it should be checked how the added function is operating inside and how the variables changes. Although these are the basic functions of a simulator, change of a request occurs in several phases of research. To meet this, we took a modular design approach which separates a visualization function, etc., from the essential computation of the ATN. The modular structure of ATN-P2P is shown in the Fig. 2.

The central box named 'ATN framework' is the framework system, a core module which executes the ATN computation. The above functions, i.e., the insertion and deletion of a node or the change in behavior of an agent and so on, are implemented on this framework. The left visualizer module is a controller of the ATN framework, which has the function of visualizing the graphical data returned from the ATN framework. The ATN framework has an API to control from outside. It is possible to control the ATN framework from not only visualizer module but also from a script and to make it deploy as a web server. On the other hand, it has a plug-in structure for gathering such information as the output in the gnuplot form of each pass state of the ATN, the variable values that nodes have for processing statistically, and fine motions in the processes of the FP and the BP, and the operations of agents. Although the visualizer module was implemented as a controller form, the data for visualization can also be collected using this plug-in function. This framework design and the modularity enhance the extendibility of the system and the independency of computation logic in the ATN, helping a researcher concentrate on the research of the mechanism of the ATN itself.

2.2 Requirement for P2P Deployment

In general, when we apply the P2P framework to some system, we have to consider the folowing issues which might become primary obstacles to the deployment:

1. Existence of shared resources.
2. Synchronicity over two or more nodes.
3. Consistency between nodes.

The existence of shared resources becomes a factor which disturbs the decentralization of processing. When a node is decentralized in the ATN, access from all of the nodes is focused on the shared resources, and there is a bottleneck in the performance.

In a platform including a higher-level manager that looks down at the whole network, synchronicity over two or more nodes is easily implemented; and yet, this is not the case in a distributed system. Simulation of the ATN requires repeated operations of the FP, the BP, and agents as one cycle (called pass), and for this reason, each distributed node needs to detect the completion of each pass.

The consistency between nodes becomes a problem, for example, when a change of the topology of the ATN is simultaneously made by two or more agents and the consistency of topology is no longer guaranteed. It may also happen that the a partial separation of the network occasionally occurs and the token flows of the data flow network are stopped.

3 Mechanisms for the P2P Platform Simulating the ATN

3.1 Flexible Resource Allocation

Since, through the repetition of passes, the ATN changes its topology dynamically and the number of nodes tends to increase with time, we cannot assign one node to one machine when performing the ATN on the machines of fixed number. To get around this problem, the ATN-P2P assigns the ATN nodes to the physical resources dynamically by dividing the node set into some groups and allocating them to the machines. One group is treated as a process, and the number of processes is taken to be changeable during the

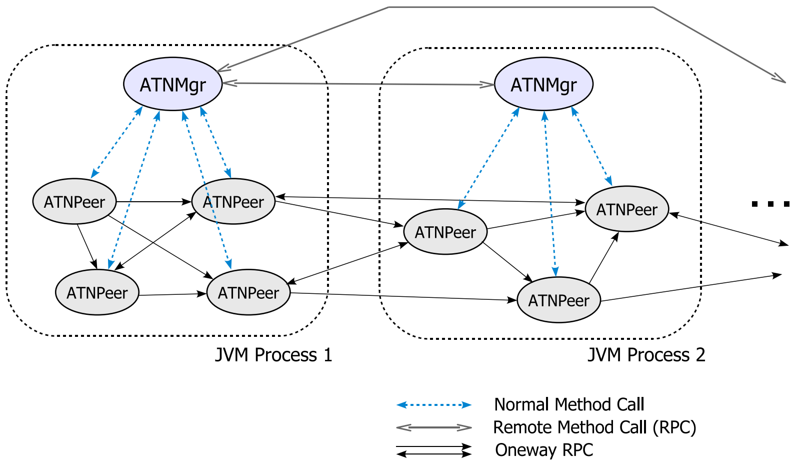


Fig. 3. Node allocations and communication patterns in the ATN-P2P

run. Furthermore, communication between processes is performed using IP addresses, irrelevant of whether it is inside or outside a machine. This makes a process boundary equivalent to a machine boundary, and assures the flexibile assignment of the processes to the machines.

Basically, we can choose arbitrarily a way for the grouping of nodes in the ATN into processes on the equipped distributed machines. For example, when the whole ATN is assigned to one process without a grouping, the ATN-P2P functions as a single simulator. When nodes are grouped one by one and each process is assigned to a machine, the ATN-P2P operates as an actual P2P network. The grouping into a process and communication of nodes are more concretely shown in the Fig. 3.

Since the ATN-P2P is implemented in Java, a process here is a JVM (Java VM) process. There exists one object called 'ATNMgr' on a JVM process. The ATNMgr manages all the nodes assigned to the process by conducting the following tasks.

– Insertion and deletion of a node and management of topology information including edges.
– Cooperation with ATNMgr(s) on other processes.
– Communication with external applications and specifying its API.

An ATNPeer is a Java object which represents a node. An outgoing edge from the node is also assigned to this ATNPeer. Communication between the ATNMgr and ATNPeer(s) is performed as follows.

– Communication between ATNPeer(s) within the same process as the ATNMgr is performed by a normal method call.
– Communication of ATNMgr(s) between different processes uses a remote procedure call (RPC).
– Communication of ATNPeer(s) uses a RPC with no returning value (called 'Oneway RPC').

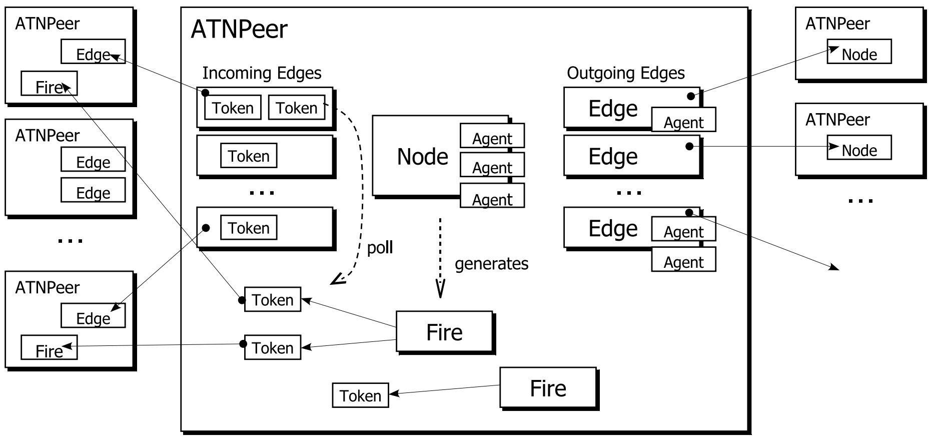


Fig. 4. Relationship of objects in ATNPeer. A 128-bit unique ID is used for the reference to Node, Edge, and Fire objects which exist in a different ATNPeer. The ATNPeer has the ID same as the Node has.

As the data-flow network used in the ATN communicate only between an adjacent node pair, so an ATNPeer exchanges such information as tokens only with its partners (adjacent ATNPeers). Moreover, in communication between ATNPeers, a communication partner's ATNPeer does not know whether the sender exists in the same JVM process. The Oneway RPC, described in the Section 4.2, is a mechanism that enables the one way message transmission in a RPC form.

The ATNMgr performs the insertion and deletion of nodes within the JVM process. It is important to decide which process of which machine the newly generated node should be assigned to. At present, the generated node is assigned to the same process as that which the parent node belongs to; however, we are able to allow an ATNPeer to migrate to another JVM process to secure good load balance between JVM processes. The problem for making rearrangement of the ATNPeers is to be solved by the cooperation of ATNMgr(s).

3.2 Object Assignment in ATNPeer

The internal structure of an ATNPeer is shown in the Fig. 4. The ATNPeer has the assigned Node object, outgoing Edge objects and Agent objects on each Node or Edge object, and Fire and Token objects generated at the time of the FP. The information on the incoming edge is also needed by the ATNPeer, but this information is made available via the reference to an Edge object which another ATNPeer holds. During the FP, if the Token objects used as the conditions for firing are prepared in the the queuing elements in the incoming Edge objects, the node fires, which causes the creation of a Fire object the ATNPeer.

3.3 Unified Description for Message Sending

In order for the ATN nodes to be allocated flexibly to physical resources, it is desirable that the communication between nodes might be expressed in a program uniformly regardless of the way of the grouping of the nodes or the assignment of the processes to the machines. The guiding principle for this unified description is:

- Eliminate a bidirectional communication.
- Always use a remote communication protocol.
- Specify the destination node with the ID.

According to the data flow model which the ATN has, we basically consider one way message transmission. The scheme of communication is unified into description supposing remote communication for both within-a-process and between-processes communication. In order to improve the performance of local communication, when the communication locality is detected, a prepared optimization procedure is always conducted. For the references to objects in another ATNPeer, the IDs are used as described in Section 3.2. This avoids the disturbance of the communication after the rearrangement takes place and the communication address (IP address etc.) of the ATNPeer changes. The above-mentioned guiding principle is realized by the Oneway RPC described in Section 4.2.

3.4 Concurrency Control of ATN

The realization of effective parallelism and concurrency is a central theme in the research of the ATN. In case of the ATN-P2P, the following two step approaches are taken.

Division of the Processes to Two or More Machines. Higher parallelism is achieved by dividing a node set into the processes on multi-core machines or processes on different machines.

Event Driven Process. Unit operations in the ATN node are classified into the firing in the FP, fire extinguishing in the BP, or operations by the agents. Since all these processes are triggered by external events and none of them use a CPU stably, we do not assign a thread to a node permanently but assign a thread to the generated event. This achieves high level concurrency in a simulation.

4 Implementation

4.1 Distinctive Feature of PIAX Transport

In the implementing of ATN-P2P, the function which the transport layer of PIAX has for the realization of the unified description of communication between nodes and the concurrency using an event driven model was used.

PIAX is a platform system for the mobile agent which operates on a P2P network. It has a framework for the nodes which constitute a P2P network in order to communicate not using physical addresses like IP addresses but using global unique IDs. This is called the 'ID/Locator separation' and it is prepared as a communication mechanism for a P2P network (or an overlay network) in the transport layer of PIAX [6].

Regarding the high level concurrency, PIAX has a mechanism for making nodes of tens of thousands of scales perform on one JVM. This is because the transport layer of PIAX is implemented as the event driven model. The feature utilized by ATN-P2P is as follows.

- Since the thread is not assigned to the node, even if it works, the nodes of tens of thousands of scales don't generate consumption of the thread.
- Make concurrent processing generated by event using a thread. Since many threads are assigned to CPU resources by JVM adaptively, they can utilize a multi-core for the maximum.
- Avoid the overhead of thread generation by thread pooling. Furthermore, the draining of thread assignment of the concurrent processing, which takes place explosively by giving thread pool restriction can be prevented by setting an upper limit on the thread pool.

4.2 Oneway RPC

For the unified description of the communication described in Section 3.3, a RPC for the one way call, named 'Oneway RPC', was implemented using the RPC function of PIAX. Although the Oneway RPC has the same call form as a RPC, since it is the one way call, neither a return value nor the exceptions which are thrown at the receiver side is returned to a caller. And since a caller does not wait for the completion of processing of the RPC, the Oneway RPC call is completed immediately.

The example of coding of the Oneway RPC is shown in the List 1.1. In this example, in an ATNPeer, in order to add outgoing edge newly, the addIncomingEdge method which adds an incoming edge of the ATNPeer is called. As preparation for treating the ATNPeer linked as a remote object, the stub of the RPC is generated by a getOnewayStub method and an addIncomingEdge method is called out as a method which the stub has. In the usual stub generation, although the IP address of a remote object is needed, the mechanism of the ID/Locator separation which PIAX has is utilized here so that the ID can be specified.

List 1.1. A sample code of Oneway RPC

```java
public void addEdge(...) {
    ...
    try {
        ATNPeerIf stub = (ATNPeerIf) getOnewayStub(dstPeerId);
        stub.addIncomingEdge(incomingIx, getId(), edge.getId());
    } catch (UndeclaredThrowableException e) {
        ...
    }
}
```

4.3 Mobility of Agents

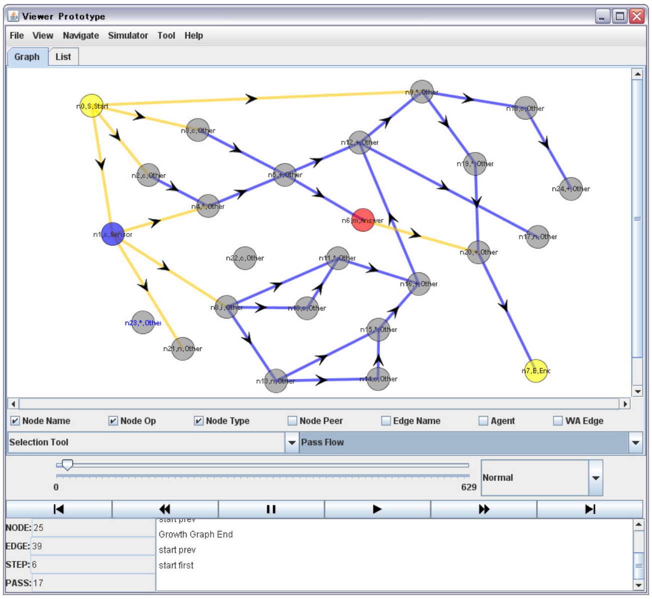
An agent mobility is a function which should be supported in the ATN. Through ATN-P2P, the agent mobility is realized using the Oneway RPC. The internal state of an Agent object is sent to the ATNPeer on the destination side using the Oneway RPC, an Agent with the same ID is generated, the internal state of the original Agent is set, and then the mobility of an agent is realized. The following method is an example which moves Agent ag to the Node whose ID is dstPeerId.

```
void moveAgent(PeerId dstPeerId, Agent ag);
```

5 Conclusion

5.1 Related Work

In the research of the data-flow network, DataRush [8] which is a simulator with the framework which can be defined by a user, a language called Stella [9] aiming at the visualization of a data flow model, and so on have been developed. And in the area studying P2P networks, there is also a system like p2psim [10] for carrying out the simulation of the operation of many nodes. Thus, tools for carrying out the simulation of the network by each area of research exist. Like ATN-P2P, a function required for a simulation can be treated integratively, and as far as the authors know, , a tool deployable as a distributed system like an actual P2P network does not exist.



[✓] Node Name [✓] Node Op [✓] Node Type [ ] Node Peer [ ] Edge Name [ ] Agent [ ] WA Edge

Selection Tool | Pass Flow

0 629 | Normal

```
NODE: 25    start prev
EDGE: 39    Growth Graph End
STEP: 6     start prev
PASS: 17    start first
```

Fig. 5. Example screenshot of the visualizer of ATN

5.2 Current Status and Future Work

Currently, the basic design of ATN-P2P has been completed and ATN-P2P is in the stage where the implementation of the foundation of a framework ended, and it cooperates with a visualizer. Fig. 5 shows the screen sample of the visualizer which is actually in operation.

In the future, a brush up of the framework, fulfillment of the plug-in function, and research and development on the function which allocates nodes to physical resources adaptively using cooperation of ATNMgr(s) are subjects to be tackled.

References

1. Suzuki, H., Ohsaki, H., Sawai, H.: An agent-based neural computational model with learning. In: Frontiers in Neuroscience, Conference Abstract: Neuroinformatics (2010), doi:10.3389/conf.fnins.2010.13.00021
2. Suzuki, H., Ohsaki, H., Sawai, H.: Algorithmically Transitive Network: A Self-organizing Data-flow Network with Learning. In: Suzuki, J., Nakano, T. (eds.) BIONETICS 2010. LNICST, vol. 87, pp. 59–73. Springer, Heidelberg (2012)
3. Sharp, J.A. (ed.): Data flow computing: Theory and practice. Ablex Publishing Corp., Norwood (1992)
4. The Message Passing Interface (MPI) standard, `http://www.mcs.anl.gov/research/projects/mpi/`
5. Yoshida, M., Okuda, T., Teranishi, Y., Harumoto, K., Shimojo, S.: PIAX: A P2P Platform for Integration of Multi-Overlay and Distributed Agent Mechanisms. IPSJ Journal 49(1), 402–413 (2008)
6. Yoshida, M., Teranishi, Y., Shimojo, S.: A Mechanism of ID/Locator Separation in Overlay Networks. IPSJ Journal 50(9), 2298–2311 (2009)
7. Gnuplot, `http://www.gnuplot.info/`
8. DataRush, `http://www.pervasivedatarush.com/`
9. Stella, `http://www.iseesystems.com/softwares/Education/StellaSoftware.aspx`
10. p2psim, `http://pdos.csail.mit.edu/p2psim/`