Managing A Cloud for Multi-agent Systems on Ad-hoc Networks

Subhajit Sidhanta
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana 70802
Email: ssidha1@tigers.lsu.edu

Supratik Mukhopadhyay
Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana 70802
Email: supratik@csc.lsu.edu

***Abstract*—We present a novel execution environment for multi-agent systems building on concepts from cloud computing and peer-to-peer networks. The novel environment can provide the computing power of a cloud for multi-agent systems in intermittently connected networks. We present the design and implementation of a prototype operating system for managing the environment. The operating system provides the user with a consistent view of a single machine, a single file system, and a unified programming model while providing elasticity and availability.**

I. Introduction

Multi-agent systems are software frameworks where autonomous intelligent agents interact with each other to achieve a computational goal. An intelligent agent is a program that receives percept streams from the environment and uses reasoning to deduce reactions to them that are actuated through actuators. Multi-agent systems have a wide range of applications in video and text analytics [7], robotic control systems [2], etc. Despite the fact that they often need to run computation-intensive algorithms to carry out complex reasoning, real time response is crucial in many mission-critical scenarios. Hence, the availability of a high performance computing infrastructure is essential for the success of such systems. Existing high performance computing infrastructures such as clusters [4] or cloud data centers [10] may not be accessible in remote areas with intermittent network connectivity such as in military battlefields, disaster scenarios, etc., where powerful expert systems are needed for analytics, resource scheduling, and other applications.

II. Design Principles

This section describes the design philosophy behind the APMAC architecture and the PPDOS operating system.

A. Design of APMAC

APMAC's design is motivated by a trade off between the non-functional requirements of a multi-agent system listed below: [1].

- Autonomy: A computation running on APMAC consists of a collection of agents running on the nodes of a network which communicate with each other to perform a set of tasks. Nodes can join and leave the network at any time. The actions which the agents perform are based on their collective knowledge of the environment as obtained through a distributed strongly consistent shared state. The shared consistent state can be maintained through a distributed key-value store that implements transactions. Agents interact with the distributed shared state through the publish-subscribe pattern. An agent embodies a function in mathematical sense whose inputs are the *monitored variables*, i.e., environment variables that the agent subscribes to and whose outputs are recorded into the *controlled variables*, i.e., environment variables that it publishes. In addition, the agent body uses *internal variables* to store intermediate results. The local state of an agent is an assignment of values to all the monitored, internal, and controlled variables. The deployment and execution of agents is managed by the PPDOS operating system. Users interact with APMAC through the PPDOS shell.
- Reactivity: The agents react in response to inputs from the environment in a timely fashion. The reactions can be published into the environment or actuated through actuators. Our results show low latency between a stimulus and its response. The local state of an agent changes in response to changes in the environment
- Deliberativity: The function embodied in an agent can be described by a set of production rules that allow reasoning about changes in the environment. The rules can be provided by a domain expert or can be learnt from data.
- Interaction: APMAC enables asynchronous communication of each agent with its environment as well as with other agents.

Apart from these features APMAC agents can be mobile, i.e., agents can migrate from one node to another in response to changes in the environment.

B. Design of PPDOS

Multiple virtual machines can be tied to a single device. The microkernel of PPDOS itself is a collection of agents that are replicated on each node. PPDOS provide the user with the view of a single machine with a single file system, a single programming model, and a Unix-like interface. Agents (both user and microkernel) communicate peer-to-peer asynchronously through a distributed key-value store (similar to a Linda's tuple-space [6]) which supports transactions. PPDOS's group management module maintains a distributed strongly consistent state of the system in the presence of joining/leaving of nodes. PPDOS allows hot-swapping of one agent with another at runtime without any disruption of service. Since PPDOS microkernel is a collection of agents, we can dynamically extend/modify the functionality of the system without disrupting its functioning. An agent can migrate from one node to another at runtime while preserving its local state. PPDOS allows interaction with the native operating systems of the nodes. Users interact with PPDOS through a command shell deploying tasks and accessing resources. Operating system services like file services, deploying and unloading agents, time, etc., are provided by a set of agents. The microkernel agents ensure that a snapshot of the state of the system is always stored in distributed key-value store.

III. Implementation

A. Agent Model

An agent running on APMAC has a predefined structure shown below (in Java).

An agent is deterministic, i.e., it embodies a function in the mathematical sense, and reactive, i.e., its state changes only in response to "events" triggered by the environment. The local variables of an agent are partitioned into monitored variables, i.e., variables through which it receives stimulus from the environment (indicated by invoking the SUBSCRIBE method in the constructor), internal variables that it uses to store intermediate values, and controlled variables that record its response to the stimulus and are published to the environment (indicated by registering to the operating system through invocation of the REGISTER method). The update methods consist of rules that allow the agent to compute responses to environmental stimuli by reasoning about it and are described by guarded commands [3] that are triggered by events in the environment.

B. Distributed Key-Value Store

For a distributed key-value store, we use Scalaris [11]. It is an open source distributed key-value store that supports transactions. Although, by default, Scalaris provides in-memory storage, through Tokyo-Cabinet [5] it can be configured to use persistent local hard disks. It implements the Paxos algorithm [9] to replicate providing ACID properties with strong consistency. It uses quorum algorithms to recover from random crashes and node restarts and can run uninterrupted even under intermittent node failures as long as a majority of the nodes in the network are available.

C. PPDOS Implementation: Multi-agent Microkernel

We have implemented PPDOS in Java. The PPDOS microkernel consists of a set of "system" agents that provides operating systems services and manage the deployment and execution of user agents. The PPDOS microkernel and the deployed user agents communicate asynchronously through the Scalaris distributed key-value store. Microkernel modules are collections of system agents that together provide a certain type of service. The microkernel modules based on their functionality can be categorized into *basic services*, *resource management services*, and *agent execution and deployment services*. We provide the algorithms implemented by the operating systems services below; for lack of space we omit the correctness proofs. The basic services are the core services provided by the microkernel that are invoked by the user agents to achieve their computation and communication goals. The Group management services are the services of the microkernel that maintain consistency in the event of nodes joining and leaving the network. A group is a dynamic collection of nodes. We have the Agent Execution Management Services that provide functions like loading,unloading and migrating agents from one node to another.

IV. Conclusions

We have presented the architecture and implementation of APMAC, a novel execution environment for running multi-agent systems on ad-hoc networks and an operating system PPDOS for managing it. PPDOS provides a single machine, strongly consistent view of the infrastructure with a single file system.

References

[1] Carla T.L.L.Silva, Jaelson Castro, Patricia Azevedo Tedesco:*Requirements for Multi-agent Systems* . WER 2003: 198-212

[2] D. Stavens, G. Hoffmann, and S. Thrun. Online speed adaptation using supervised learning for high-speed, off-road autonomous driving. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, 2007.

[3] Dijkstra, Edsger W. "EWD472: Guarded commands, non-determinacy and formal. derivation of programs." (PDF). Retrieved February 29, 2012.

[4] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003. ISBN: 0-470-85319-0

[5] fallabs, [online] 2012, http://fallabs.com/tokyocabinet/ (Accessed: 28 February 2012)

[6] Gelernter, David. "Generative communication in Linda". ACM Transactions on Programming Languages and Systems, volume 7, number 1, January 1985.

[7] Harold Trease, Tim Carlson, Ryan Moony, Robert Farber, and Lynn Trease. 2007. "Unstructured data analysis of streaming video using parallel, high-throughput algorithms". In Proceedings of the Ninth IASTED International Conference on Signal and Image Processing (SIP '07), Rui J. P. de Figueiredo (Ed.). ACTA Press, Anaheim, CA, USA, 305-310.

[8] Hastings, W.K. (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". Biometrika 57 (1): 97Ű109. doi:10.1093/biomet/57.1.97. JSTOR 2334940. Zbl 0219.65008.

[9] Lamport, Leslie (May 1998). "The Part-Time Parliament". ACM Transactions on Computer Systems 16 (2): 133Ű169. doi:10.1145/279227.279229. Retrieved 2012-03-02.

[10] Pepitone Julianne, "The NIST definitionn of Cloud Computing," (NIST), [online] 2011, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (Accessed: 16 February 2012)

[11] Thorsten Schütt , Florian Schintke , Alexander Reinefeld, Scalaris: reliable transactional p2p key/value store, Proceedings of the 7th ACM SIGPLAN workshop on ERLANG, September 27-27, 2008, Victoria, BC, Canada [doi 10.1145/1411273.1411280]

[12] Vmware.com, [online] 2012, http://www.vmware.com/pdf/virtualization.pdf (Accessed: 27 February 2012)

[13] Wikipedia, [online] 2012, http://en.wikipedia.org/wiki/Apache_Hadoop (Accessed: 28 February 2012)