Distributed State Machines: A Declarative Framework for the Management of Distributed Systems

Jorge Lobo, David Wood, Dinesh Verma and Seraphin Calo

IBM Research
Hawthorne, NY, USA
jlobo,dawood,dverma,scalo@us.ibm.com

Abstract— In this paper we describe the implementation of a declarative framework to support the development of distributed management applications. The framework is based on an extension of declarative networking, an asynchronous computational model that uses recursive SQL as its foundation and has been successfully used for the implementation of networking protocols.

I. INTRODUCTION

Over the past year and one half we have been building on an infrastructure to support the development of networking applications. The goal is to relieve the application developer of tending to the details of the communication and control.

Distributed decentralized applications that are robust to changing relationships and topologies, with no single points of failure are difficult to write and we would like to abstract out as much of the networking as possible. We take a data-driven model of computation in which computational application nodes communicate with each other through the exchange of data to solve a distributed problem. Data is collected locally and shared with others as defined by the application. Changes in local data shared with other peer application nodes can trigger those peers to take appropriate action in response to these changes. The framework is composed of three layers. The lowest layer manages topology and communications. The second layer supports the concept of shared data including how data is shared and queries over remote shared data. The third layer is related to the nature of the data that can be shared, and provides different models for data sharing. We will briefly discuss two models, one based on serializable Java objects and conditions on these objects and the other based on relational databases and conditions described in an SQL-based language.

II. DSM ARCHITECTURE

The Distributed State Machine architecture assumes a simple model of a distributed set of application nodes, each maintaining its own local data, and collaborating to solve a common problem by sharing data as necessary. An application will be distributed among different nodes and communication between nodes will happen through the Distributed State machines.

Instead of directly addressing nodes, an application uses network topology relationships such as parent, neighbor, reachable, etc., to address nodes. These features enable, for example, the implementation of routing algorithms in a mobile ad hoc network through the sharing of neighbors and known routes with node neighbors. With this framework in mind, the primary goals of the architecture are to: 1) provide topology-based addressing; 2) support dynamic networks; and 3) allow the efficient and intuitive sharing of data.

The high level architecture of a DSM node is depicted in Figure 1. A single hop broadcast/multicast algorithm is implemented within each node with both nodes and routes assigned an expiration time. This provides reachability and neighbor information about other nodes and also allows the network to partition and adapt to the changing set of connected nodes. Logical topology relationships such as parent and child are defined by the application through a platform configuration.

Shared data is assigned a unique name defined by the application. The data is then made available to peers via queries that reference one or more named data objects. Queries are either synchronous or continuous and are addressed to a set of nodes with one or more topological relationships (i.e. neighbor, parent, etc.). Leased continuous queries may specify a condition that must be met before results are returned. Conditions may be defined on a pairwise (peer-to-peer) basis or as an aggregation of values across multiple nodes in a topological relationship.

III. SHARING JAVA OBJECTS

The implementation for sharing serializable Java objects uses a hash table to map the names associated with shared data to the shared Java instance data. Support for queries based on the names of shared items is supported and more complex queries, such as field values, can easily be created. A powerful aspect of this implementation is the availability of the Java Expression Language JEXL [8] based expression

Figure 1 Architecture of a DSM node
to define the conditions used in continuous queries. The JEXL run-time is provided a set of objects over which expressions can be defined on both local, and paired or aggregated remote data. Examples using data shared with the name resources follow:

- `Local.get("resources").cpuUtilization < Peer.get("resources").cpuUtilization`
- `Local.get("resources").cpuCount() < Average.get("resources","cpuCount()")`

These operate on a shared Java object containing 1) a field named cpuUtilization; and 2) a method cpuCount(). The first condition is true whenever the cpu utilization of a peer node is larger than the CPU utilization of the current node. The second condition will become true whenever the average number of CPUs in all the peer nodes is larger than the number of CPUs in the local node. All the communication required among the nodes to compute these conditions is transparent to the application developer.

IV. SHARING RELATIONAL TABLES

The computation based on the relational model is very simple. Each node runs an input/output state machine. A state in the machine is represented by set of relational tables. State transitions (i.e. table updates) occur when the state machine receives inputs either from applications running in the node or from other state machines running in other nodes (i.e. communications between nodes only happens through the state machines). All inputs are in the form of named tuples (i.e. relational table rows). A state change can also produce an output in the form of tuples, all output tuples are accompanied with a destination, i.e. the location of another state machine which will asynchronously receive the tuple. A description of the high-level declarative rule language [4] supporting this model is beyond the scope of this paper. We have used this to implement a preferential routing algorithm based on the sharing of opinions and observations of nodes in a coalition network. The framework enabled quick and easy adjustments to the sharing and rating aggregation algorithm.

V. RELATED WORK AND FINAL REMARKS

Our work on distributed state machines is an extension to the concept of declarative networking introduced in [5]. Declarative networking platforms have been developed specifically to support the declarative implementation (Datalog based) of routing protocols. However, the computational model behind declarative networking is directed towards the definition of a single state: a stable state that the system is intended to reach. State transition happens but they are supposed to be hidden from the application developer. In our implementation we follow the suggestion made in [1] to explicitly represent state. There are important semantic differences between our implementation and the implementation in [1] but details of those differences are outside the scope of our presentation. Details can be found in [4].

We are using our framework to build applications that provide distributed execution and control of video analytics in a surveillance context, tracking of assets in a sensor field, fault management and resiliency in networked appliances, and for management of resources in a data center environment. The framework can be used to simplify the development of many new applications, primarily in the field of network and systems management, and in processing of sensor network information streams.

An important aspect of the declarative approach is the amenability for analysis. We are developing a toolkit for debugging and analysis. We currently have a basic set of analysis tools [6].

ACKNOWLEDGMENT

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions are those of the authors and do not represent the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding copyright notation.

REFERENCES