1 Introduction

Declarative Networking. Design and implementation of network protocols often give rise to some hard tradeoffs — between extensibility and flexibility on one hand, and robustness and efficiency on the other. Today’s Internet level routing and network management protocols, though robust and extensible, are not easily changeable to meet the demands of new types of applications. In the area of network security, an ever increasing number of security protocols and devices makes it difficult to detect security breaches due to not only errors in the implementations of those protocols but also conflicts between different protocols. In recent years Declarative Networking has drawn much interests in network research community due to its capability of dealing with these issues efficiently. Declarative Networking is the application of database query techniques in the domain of networking\cite{7}\cite{6}\cite{5}. Recursive query language can represent the network protocol in a compact manner which makes it more extensible and less error prone.

Generally in the networking area, Declarative Networking is applied to make routing algorithms more extensible and flexible while maintaining robustness and efficiency. By extensible we mean routing algorithms can be changed with least or no overhead to accommodate the needs of new applications. A most notable advantage of the Declarative Networking paradigm is that it can be used to handle both routing in networks as well as resource allocation and scheduling very efficiently. Our goal is to exploit this benefit in the domain of Grid Computing.

Grid Computing. Grid computing has been devised basically to meet the need of eScience community to deal with computations involving very large data sets in domains such as astrophysics, particle physics, climate modeling and fluid dynamics. Consumer-oriented applications can also use Grid Computing infrastructure. For example, in high-definition (HD) video editing, applying effects requires one or multiple operations per pixel, or en/decoding these high resolution and high frame rate image streams quickly which leads to a non-negligible amount of processing which already is challenging on todays commodity computers.

We apply Declarative Networking techniques in the area of Grid Computing which, as far we know, has not been explored yet. In this computing environment, when a client request arrives for a specific task, the controller (centralized or distributed) reserves computing and storage resources for that task in multiple servers so that computation can be done in parallel. At the same time the controller needs to route the flow of data between the client and servers and among the servers. Hence it is a joint scheduling of computing
and communication resources. So far Declarative Networking has only been applied for routing, but joint routing and resources scheduling is one area that has potential to be benefitted from Declarative Networking.

Outline. The remainder of this report is organized as follows. Section 2 gives an account of the related work in this area. An introduction to the declarative languages Datalog and NDlog is given in Section 3. Our approach of application of Declarative Networking to Grid Computing is detailed in Section 4. Finally Section 5 concludes the report.

2 Related work

Declarative Routing. Some of the early works in this area focus on the intuition behind declarative programming of networks, and its root in Datalog[7][5]. NDlog, an extension to traditional Datalog has been presented in the literature as a Network Protocol Language. NDlog extends datalog to include the semantics of long-running queries over network state. In addition, several ramifications of the application of the Declarative Networking paradigm have been discussed like automatic protocol optimization and hybridization, program checking and debugging[6].

Declarative Network Operation and Management. The declarative networking paradigm has also been applied in the field of network management to ease complicated and error-prone operations. The lack of automation in network management is mainly due to a lack of programmability at the right level of abstraction. To address this issue, DECOR—a database-oriented declarative framework—has been presented [3]. Using DECOR, network management operations can be represented as a series of transactional database queries.

Secure Distributed Systems. Much work has been done in the field of network security using Declarative Networking. SeNDlog, a logic-based language, has been proposed for managing access control in distributed systems [1]. There has also been efforts to investigate security challenges in cloud data management using the Declarative Secure Distributed Systems platform [9]. Some of the interesting challenges in this area include secure distributed data processing, end-to-end query result verification, and cross-user trust policy management. The SeNDlog language extends NDlog with basic security constructs for the implementation of secure distributed systems. These are further enhanced with type checking and meta-programmability in LBTrust system [8] for supporting different forms of encryption, delegation, for distributed trust management.

Datacenter Programming. In the setting of cloud computing, the BOOM project is exploring the use of declarative languages. The sequential programming models that are provided by current cloud platforms are not suitable for inherently distributed resources. Overlog has been used to illustrate the benefits of declarative programming in a cloud; it acts as the basis for a simplified and enhanced reimplementation of a cloud-based analytics stack: the Hadoop File System (HDFS) and MapReduce infrastructure[2]. High level Overlog specifications easily enable developers to quickly add sophisticated distributed features to networked systems via data partitioning, and implementations of new scheduling protocols and query processing strategies.

Other Miscellaneous Applications. Declarative Networking has also been successfully adopted in the areas of mobility-based overlays, adaptively hybridized mobile ad-hoc networks, overlay network composition, sensor networking, fault-tolerant protocols, network configuration, replicated filesystems, distributed machine learning algorithms, and robotics.
3 Languages: Datalog and NDlog

Loo et al. [7] have formally defined the Network Datalog (NDlog) language for declarative networking. The NDlog language is based on extensions to traditional Datalog, a well-known recursive query language traditionally designed for querying graph-structured data in a centralized database.

**Datalog.** A Datalog rule has the form \( p : \neg q_1, q_2, ..., q_n \), which can be read informally as \( q_1 \) and \( q_2 \) and ... and \( q_n \) implies \( p \). \( p \) is the head of the rule, and \( q_1, q_2, ..., q_n \) is a list of literals that constitutes the body of the rule. Literals are either predicates over fields (variables and constants), or function symbols applied to fields. The rules can refer to each other in a cyclic fashion to express recursion. The order in which the rules are presented in a program is semantically immaterial. The commas separating the predicates in a rule are logical conjuncts (AND); the order in which predicates appear in a rule body also has no semantic significance, though most implementations employ a left-to-right execution strategy. The query specifies the output of interest.

Most implementations of Datalog enhance it with a limited set of function calls (which start with “f” in standard syntax), including boolean predicates and arithmetic computations. Aggregate constructs are represented as functions with field variables within angle brackets (<>).

**Towards Network Datalog.** NDlog tries to integrate networking and logic, which is unique from the perspective of both domains. As a network protocol language, it is notable for the absence of any communication primitives like “send” or “receive”; instead, communication is implicit in a simple high-level specification of data partitioning. In comparison to traditional logic languages, it is enhanced to capture typical network realities including distribution, link-layer constraints on communication and soft state semantics.

The properties of NDlog are as follows.

- **Distributed Computations:** Since network protocols are typically computations over distributed network state, one of the important requirements of NDlog is the ability to support rules that express distributed computations. NDlog builds upon traditional Datalog by providing control over the storage location of tuples explicitly in the syntax via location specifiers. Each location specifier is a field within a predicate that dictates the partitioning of the table. In standard implementation, a “@” symbol is prepended to a single field denoting the location specifier. Each tuple generated is stored at the address determined by its location specifier. [6]

- **Link-restricted communications:** In order to send a message in a low-level network, there needs to be a link between the sender and receiver. This is not a natural construct in Datalog. Hence, to model physical networking components where full connectivity is not always available, NDlog provides syntactic restrictions that can be used to ensure that rule execution results in communication only among nodes that are physically connected. This is syntactically achieved with the use of the special #link predicate in all NDlog programs. [4]

- **Soft-state data and rules:** In typical network protocols, the generated network state is maintained as soft-state data. In the soft state storage model, stored data have a lifetime or time-to-live (TTL), and are deleted when the lifetime has expired. The soft state storage model requires periodic communication to refresh network state. Soft state is often favored in networking implementations because in a very simple manner it provides well-defined eventual consistency semantics. Intuitively, periodic refreshes to network state ensure that the eventual values are obtained even if there are transient errors such as reordered messages, node disconnection or link failures. While soft state is useful for maintaining distributed state, we also make extensive use of traditional “hard-state” data with infinite lifetimes for storing persistent counters, local machine state and archival logs. [4]

- **Incremental maintenance of network state:** In practice, most network protocols are executed over a long period of time, and the protocol incrementally updates and repairs routing tables as
the underlying network changes (link failures, node departures, etc). To better map into practical networking scenarios, one key distinction that differentiates the execution of NDlog from earlier work in Datalog is the support for continuous rule execution and results materialization, where all tuples derived from NDlog rules are materialized and incrementally updated as the underlying network changes. As in network protocols, such incremental maintenance is required both for timely updates and for avoiding the overhead of recomputing all routing tables “from scratch” whenever there are changes to the underlying network. [4]

An example. We show an example NDlog program that finds the reachability of nodes in a graph. We say that in a graph, a node $A$ is reachable from a node $B$ if there is a path between the two nodes in the graph.

The reachability between nodes in a graph can be specified by the following NDlog program:

R1: \texttt{reachable}(\@S,D) :- \texttt{link}(\@S,D)

R2: \texttt{reachable}(\@S,D) :- \texttt{link}(\@S,Z),\texttt{reachable}(\@Z,D)

Q1: \texttt{reachable}(\@M,N)

Q2: \texttt{reachable}(\@a,N)

The program has two rules (which we label R1 and R2). The rule R1 derives reachable tuples directly from the link tuples. It says that a node $D$ is reachable from a node $S$ if there is an edge between nodes $S$ and $D$. The rule R2 recursively produces reachable tuples by matching the $D$ fields of existing \texttt{link} tuples to the $S$ fields of previously computed reachable tuples. The matching is expressed using the repeated $Z$ variable in \texttt{link}(\@S,Z) and \texttt{reachable}(\@Z,D) in rule R2. Rule R2 can be read as: a node $D$ is reachable from a node $S$ if there is a link between $S$ and a node $Z$, and $D$ is reachable from $Z$.

Q1 and Q2 are two sample queries. Execution of the query Q1 returns the reachable tuples for every node pair in the graph i.e. it returns the all-pairs reachability. Execution of query Q2 returns all the tuples stored at node $a$ i.e. all the nodes that are reachable from node $a$.

Figure 1: Nodes in the graph are running the reachability program. We only show newly derived tuples at each iteration.

We next step through an execution of the reachability NDlog program to illustrate derivation and communication of tuples as the program is computed. We use the example graph in Figure 1. We only focus on the high level understanding of the data movement in the network (graph) during query processing. Commu-
unication can be described in synchronized iterations, where at each iteration, each network node generates reachable tuples where the distance between S and D is increasing by 1. The tuples are then propagated to neighbor nodes along bi-directional network links (shown in bi-directional dashed arrows in the figure).

Initially all the nodes have the link tuples only, which store information about the links (edges) in the graph. In the first iteration, all nodes initialize their local reachable tuples using the rule R1. In the second iteration, using rule R2, each node takes the input reachable tuples generated in the previous iteration and computes new reachable tuples, which are then propagated to the neighbours. For example reachable(@a, c) is generated at node b using reachable(@b, c) from the first iteration, and propagated to node a. In this way after sufficient number of iterations all the reachable tuples are populated.

**Execution of NDlog queries.** In their implementation [6], Loo et al. explain that the runtime execution of NDlog programs differs from the traditional implementation patterns for both network protocols and database queries. Network protocol implementations often center around local state machines that emit messages, triggering state transitions at other state machines. In contrast, the runtime system for NDlog is a distributed dataflow execution engine, similar in spirit to those developed for parallel database systems, and also in recent parallel map-reduce implementations.

The authors describe the four main phases of the query execution system – Centralized Plan Generation, Distributed Plan Generation, Semi-naïve Evaluation and Incremental Maintenance. We skip the details here, but they are interesting from a database research perspective.

4 **Our Approach**

Declarative Networking has been applied in many areas such as intra and inter domain routing protocols (e.g., Distant Vector Routing, BGP), Overlay protocols (e.g., Chord DHT), Distributed systems (e.g., Gnutella), Network management configuration and security verification and so on. How this technique can be used to express Grid computing protocols is an interesting problem. The ultimate goal of declarative networking research is to represent each and every protocols used in today’s network using declarative language to make it more flexible and extensible so that the network can easily be changed to meet the demands of new types of applications in the coming days. We want to contribute by exploring the possibilities of applying this technique in the design of Grid Computing. Grid computing has been devised basically to meet the need of eScience community to deal with computations involving very large data sets in domains such as astrophysics, climate modeling and fluid dynamics. In a grid, a set of clusters are interconnected with high bandwidth transmission lines. This set of clusters forms a common pool of computing and storage resources for end users. We will look at the grid dimensioning problem which means figuring out how to schedule computing and storage resources for an incoming request and routing the request efficiently through the grid so that it optimizes resource usage. Declarative programming can be used to describe all these resources and usage constraints in a unified manner.

5 **Conclusions**
References


