Declarative Constraint Optimization in Distributed Systems

Changbin Liu
University of Pennsylvania

Boon Thau Loo
University of Pennsylvania, boonloo@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_papers
Part of the Computer Sciences Commons

Recommended Citation


This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_papers/689
For more information, please contact repository@pobox.upenn.edu.
Declarative Constraint Optimization in Distributed Systems

Abstract
In distributed systems management, operators often have to configure system parameters that optimize performance objectives given constraints in the deployment environment. This position paper presents our recent work on a declarative optimization platform that enables constraint optimization problems (COP) to be declaratively specified and incrementally executed in distributed systems. Traditional approaches in implementing COPs use imperative languages like C++ [2] or Java [1]. This often results in multi-thousand lines of code, that are difficult to maintain and customize. Moreover, due to scalability issues and management constraints imposed across administrative domains, it is often necessary to execute a COP in a distributed setting, where multiple local solvers coordinate with each other and each one handles a portion of the whole problem to together achieve a global objective. Central to our optimization platform is the integration of a declarative networking [9] engine with an off-the-shelf constraint solver [2]. We have applied our platform to two use cases. First, in mesh networks, policies on wireless channel selection [7, 6] are declaratively specified and optimized, in order to reduce network interference and maximize throughput, while not violating constraints such as refraining from channels owned exclusively by the primary users. Second, in automated cloud resource orchestration [8], we use our optimization framework to declaratively control the creation, management, manipulation and decommissioning of cloud resources, in order to realize customer requests, while conforming to operational objectives of the cloud service providers at the same time. Beyond these two use cases, we envision our platform has a wide-range of potential applications, for example, optimizing distributed systems for load balancing, robust routing, scheduling, and security.

Disciplines
Computer Sciences

Comments

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/689
Declarative Constraint Optimization in Distributed Systems

Changbin Liu  
University of Pennsylvania  
3330 Walnut St, Philadelphia  
PA, USA  
changbl@cis.upenn.edu

Boon Thau Loo  
University of Pennsylvania  
3330 Walnut St, Philadelphia  
PA, USA  
boonloo@cis.upenn.edu

1. INTRODUCTION

In distributed systems management, operators often have to configure system parameters that optimize performance objectives given constraints in the deployment environment. This position paper presents our recent work on a declarative optimization platform that enables constraint optimization problems (COP) to be declaratively specified and incrementally executed in distributed systems.

Traditional approaches in implementing COPs use imperative languages like C++ [2] or Java [1]. This often results in multi-thousand lines of code, that are difficult to maintain and customize. Moreover, due to scalability issues and management constraints imposed across administrative domains, it is often necessary to execute a COP in a distributed setting, where multiple local solvers coordinate with each other and each one handles a portion of the whole problem to together achieve a global objective.

Central to our optimization platform is the integration of a declarative networking [9] engine with an off-the-shelf constraint solver [2]. We have applied our platform to two use cases. First, in mesh networks, policies on wireless channel selection [2] are declaratively specified and optimized, in order to reduce network interference and maximize throughput, while not violating constraints such as refraining from channels owned exclusively by the primary users. Second, in automated cloud resource orchestration [8], we use our optimization framework to declaratively control the creation, management, manipulation and decommissioning of cloud resources, in order to realize customer requests, while conforming to operational objectives of the cloud service providers at the same time. Beyond these two use cases, we envision our platform has a wide-range of potential applications, for example, optimizing distributed systems for load balancing, robust routing, scheduling, and security.

2. DECLARATIVE LANGUAGE

Our optimization platform uses the Colog declarative policy language, which allows operators to concisely model distributed system resources and formulate management decisions as declarative programs with specified goals and constraints. Compared to traditional imperative alternatives, Colog results in orders of magnitude reduction in code size, and is easier to understand, debug and extend.

Given the space constraints, we refer the reader to [7, 6, 8] for details on the language, complete examples, and use cases. Here, we present some high level intuitions on the language.

Colog declarative policy language is based on Datalog, a recursive query language used in the database community for querying graphs. Our choice of Datalog as a basis for Colog is driven by Datalog’s conciseness in specifying dependencies among system states, especially distributed system states that exhibit recursive properties. Its root in logic provides a convenient mechanism for Colog to extend traditional Datalog with constructs expressing COP formulations in the form of policy goals and constraints. Moreover, there exists a distributed Datalog engine used in declarative networking that facilitate distributed COP computations. Colog specifications are compiled into execution plans executed by a distributed query engine integrated with constraint solving modules.

In Colog, regular Datalog rules are used to generate intermediate tables used by the solver. This is specified as regular Datalog rules of the form \( p :- q_1, q_2, \ldots, q_n \), resulting in the derivation of \( p \), whenever the rule body \( (q_1 \text{ and } q_2 \text{ and } \ldots \text{ and } q_n) \) is true. We adopt standard Datalog terminology, and refer to each term within a rule (e.g. \( q_1 \text{ and } q_2 \)) as a predicate, and the corresponding derivation obtained (e.g. \( p \)) during rule body execution is referred to as tuples.

Language extensions. Two reserved keywords goal and var specify the optimization goal and variables used by the constraint solver, respectively. Constraint rules of the form \( p \rightarrow q_1, q_2, \ldots, q_n \), denotes the logical meaning that whenever \( p \) is true, then the rule body \( (q_1 \text{ and } q_2 \text{ and } \ldots \text{ and } q_n) \) must also be true to satisfy the constraint. Unlike a Datalog rule, which derives new values for a predicate, a constraint restricts a predicate’s allowed values, hence representing an invariant that must be maintained at all times. These are used by the solver to limit the search space when computing the optimization goal. Using Colog, it is easy to customize policies simply by modifying the goals, constraints, and adding additional derivation rules.

Distributed COP. Colog can be executed in a distributed setting. At a high level, multiple solver nodes execute a local COP, and then iteratively exchange COP results with neighboring nodes until a stopping condition is reached. The distributed COP program is written using distributed variant of Datalog used in declarative networking, where a location specifier \( @ \) denotes the source location of each corresponding tuple. This allows us to write rules where the input data spans multiple nodes, a convenient language construct for formulating distributed optimizations.
3. Compilation and Execution

To execute Colog programs in a distributed setting, COPE integrates Gecode [2], an off-the-shelf constraint solver and the RapidNet declarative networking engine [3] for communicating policy decisions among different solver nodes.

RapidNet was originally designed as a platform for executing declarative networks [4]. We adopted its usage in order to leverage its distributed Datalog engine. This allows us to execute the derivation rules in Colog programs using standard query processing techniques involving database operators, such as joins (variable matching in rule body), aggregation (e.g. SUM, MAX), selection filters, and rule head renaming.

Whenever solving a COP, Colog programs are compiled into executable code in RapidNet, which invokes Gecode's high-performance constraint solving modules. Our compilation process maps Colog's goal, var, and constraints into equivalent COP primitives in Gecode. These modules are invoked either as a one-time program, periodically (via periodic timer events generated from Colog rules), or in a continuous fashion triggered by incremental maintenance [5] as the body predicates are updated.

Gecode adopts the standard branch-and-bound searching approach to solve the optimization while exploring the space of variables under constraints. In addition to these constraints, rules that use solver results as input are rewritten into constraints to further prune the search space.

One of the interesting aspects of Colog, from a query processing standpoint, is our integration of RapidNet (an incremental bottom-up distributed Datalog evaluation engine) and Gecode (a top-down goal-oriented constraint solver). This integration allows us to implement a distributed solver that can perform incremental and distributed constraint optimizations.

To execute distributed COPs, Colog uses RapidNet for executing distributed Datalog rules, which already provides a runtime environment for implementing these rules. At a high level, each distributed rule or constraint (with multiple distinct location specifiers) is rewritten using a localization rewrite [6] step. This transformation results in rules bodies that can be executed locally, and rule heads that can be derived and sent across nodes. The beauty of this rewrite is that even if the original program expresses distributed properties and constraints, this rewrite process will realize multiple local COP operations at different nodes, and have the output of COP operations via derivations sent across nodes.

4. Representative Use Cases

Wireless Channel Selection [7, 6]. Our first example is wireless channel selection in mesh networks. Colog is used to address the problem of assigning wireless channels to multi-radio multi-channel nodes to reduce interference based on the one-hop interference model [12]. In this model, any two adjacent links are considered to interfere with each other if they both use channels whose frequency bands are closer than a certain threshold. This is equivalent to classic graph coloring problem [6]. Moreover, Colog can be easily modified to support the more complex two-hop interference model [13], which is often considered a more accurate measurement of interference in wireless deployments such as IEEE 802.11. This model considers interference that results from any two links using similar channels within two hops of each other. In addition, COPE can flexibly declare more constraints, e.g. impose regional policies on spectrum usage, avoid channels with low SNR, ensure channel diversity along each path.

Cloud Resource Orchestration [8]. Our second example is cloud resource orchestration in the Follow-the-Sun [11] cloud service, which aims to migrate VMs across geographical distributed data centers based on customer dynamics. Here, the geographic location of the primary workload (i.e., the majority of customers using the cloud service) derives demand shifts during the course of a day, and it is beneficial for these workload drivers to be in close proximity to the resources they operate on. The migration decision process has to occur in real-time on a live deployment with minimal disruption to existing services. In this scenario, the cloud infrastructure service aims to optimize for two parties: enable service consolidation (for providers) to reduce operating costs, and improve application performance (for customers), while ensuring that customer SLAs of web services (e.g. defined in terms of the average end-to-end experienced latency of user requests) are met. In addition, they may be performed to reduce inter-data center communication overhead [13]. Since data centers in this scenario belong to different cloud providers (similar to federated cloud [4]), Colog are executed in a distributed setting, where each solver node is responsible for controlling resources within their data center.

5. References