Constraint-Based Allocation of Cloud Resources to Maximize Mission Effectiveness

Mark Boddy*
Adventium Labs
111 Third Avenue South, Suite 100
Minneapolis, MN 55401 USA
mark.boddy@adventiumlabs.com

Abstract

We are concerned with the problem of optimizing network resource allocations to mission tasks. The model includes unreliable network assets, multiple mission tasks and phases, and the possibility of over-provisioning one or more tasks as a means of increasing the likelihood of task success. In this paper, we describe an implemented approach to optimizing network resources so as to optimize the expected utility of the mission. This differs significantly from previous work on cloud and network management, where the objective was to optimize some operational measure of the network itself, rather than the effect of network failures on a specific task. The work described here is preliminary: we describe the problem and the approach, define an architecture, and present the current state of the implementation.

Introduction

We are concerned with the problem of optimizing network resource allocations to mission tasks. The model includes unreliable network assets, multiple mission tasks and phases, and the possibility of over-provisioning one or more tasks as a means of increasing the likelihood of task success. In this paper, we describe an implemented approach to optimizing network resources so as to optimize the expected utility of the mission. This differs significantly from previous work on cloud and network management, where the objective was to optimize some operational measure of the network itself, rather than the effect of network failures on a specific task. The work described here is preliminary: we describe the problem and the approach, define an architecture, and present the current state of the implementation.

Optimizing cloud resource allocations to mission tasks with unreliable network assets and over-provisioning requires active management of cloud resources, using an explicit model of the effect of resource failures on the success of mission tasks, and of the dependence of the overall mission on those tasks. The Allocation of Missions Built on Resource Optimization (AMBORO) system[1] provides a flexible, adaptable, and effective means of optimizing the *expected utility* of cloud resource allocations to mission tasks, given estimates of the reliability of those resources. AMBORO provides a stable, extensible, scalable platform for modeling and solving cloud management problems involving uncertain information.

Using a uniform, constraint-based representation for modeling mission tasks, cloud resources and the current set of assignments of resources to tasks provides several benefits, including easy extension of the model to reflect new types of missions, tasks, network components, and operational requirements such as Quality of Service (QoS) guarantees, as well as providing access to a very wide variety of implemented solvers. Other advantages to a constraint-based approach include the ability to define partial solutions, culprit identification in the case of infeasibilities or other conflicts, incremental solutions as additional tasks are added, and the use of local search methods.

In this paper, we describe AMBORO, including its architecture, previous work on which it builds, the form of the input models, and the optimization problem formulation and solution process. We then conclude by reviewing the current state of the system and some planned, near-term future work.

Related Work

In some sense, there is little or no current work in the area being addressed here, because the problem as defined involves modeling and optimization of brand new capabilities. For example, trading off the costs versus the benefits of various forms of Moving Target Defense (MTD) would requires the existence of models (and the experience to populate those models), describing their costs and effectiveness against specific forms of network attack. This is very much an emerging area of research. Mapping from specific measures of system and network performance such as latency or throughput to their effect on overall mission success is another area in which there is not much work to compare to.

However, there is a extensive body of current and recent work on mapping tasks onto the resources available in a

*This work was supported by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA), under contract from AFRL, contract # FA8750-11-C-0265. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited

[1] Amboro is a cloud forest in Bolivia.

distributed system, whether it is called a cloud, a grid, or just a network. Some of this work employs control-theoretic approaches to adaptive load-balancing for virtual machines across physical servers, in the presence of other resource constraints (Hyser et al. 2007; Wood et al. 2007), or optimizing the use of resources such as power, again under constraints derived from the tasks being supported (Wang and Wang 2010; Padal et al. 2007). Other research makes use of game theory and other methods for decision-making under uncertainty to allocate unreliable network assets to computational tasks (Sarmenta 2002; Sonnek, Chandra, and Weissman 2007), for example using statistics on past behavior to balance task throughput and reliability. Singh, Korupolu and Mohapatra (Singh, Korupolu, and Mohapatra 2008) describe a sophisticated online algorithm for hierarchical, multi-dimensional load-balancing across a distributed system, based on an algorithm for the multi-dimensional knapsack problem. In their conclusion, they discuss extending this work based on statistical information similar to that in the work cited just above.

What distinguishes our work is the need to represent and reason about expectations, trade offs involving risk and resource cost for deploying monitoring and active defenses, and the effect of network asset compromises on measures of mission effectiveness.

Finally, there are simulation-based tools that can be used to configure network resources, such as Opnet's IT Guru Network Planner [2]. Simulation and sampling approaches can be used to evaluate alternative network configurations, including the behavior of those configurations under adverse conditions such as network outages. These tools also provide or support static analysis tools, such as Opnet's "Survivability Report," and evaluation against regulatory policies and vendor "best practices." What they do not do is provide a means to find a good configuration in the first place, or to modify the current configuration as the situation changes.

CARINAE

The departure point for AMBORO is a system called Cyber Architecture Reasoner Inferring Network and Application Environments™ (CARINAE), described in (Michalowski, Boddy, and Carpenter 2010). As shown in Figure 1, CARINAE is a model-based, trust-driven tool for configuring defensive cyber operations. Given a network model and information regarding current and planned network operations in support of both missions and network defense, CARINAE provides network operators with the means to detect and resolve resource conflicts in network cyber-defense operations.

With CARINAE, network architects and operators can predict and resolve multiple scalability issues, including physical and logical network topologies, defended application resource loads, and defensive application architecture and resource requirements. Focused on large, service-oriented net-centric enterprise systems, CARINAE leverages constraint-based reasoning and open source, industry standard tools to create a robust analytical architecture that

[2]http://www.opnet.com/solutions/network_performance/index.html

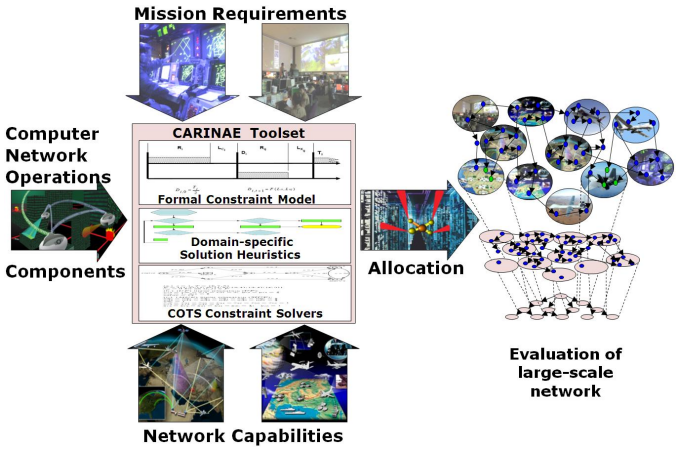

Figure 1: Cyber Architecture Reasoner Inferring Network and Application Environments™ (CARINAE)

can analyze the interactions between network configurations and mission requirements for large-scale defensive cyber applications (Michalowski, Boddy, and Carpenter 2010), (Haigh, Harp, and Payne 2010). CARINAE provides bandwidth, memory, and computational performance guarantees for large networks supporting diverse operational missions and defensive applications. Based on a collection of innovative modeling and algorithmic optimizations, CARINAE has been employed to analyze networks consisting of up to 1,000,000 nodes, with solution times typically well under a minute.

CARINAE's constraint-based model supports easy extension to a wide diversity of network topologies, node configurations, and mission requirements. The model supports the representation of hierarchies of processing nodes and hierarchical resource usage, supporting server-based virtualization and a hierarchy of network services, as well as network links tagged with various attributes, supporting requirements such as encrypted links. However, CARINAE does not support an explicit mapping from tasks to resource demands: the demands are provided directly. Nor does CARINAE's model or solution engines support reasoning about uncertainties such as the trustworthiness of a given node, or the likelihood of task or mission success.

In CARINAE, we employed the Coin-OR Linear Programming solver, as well as the Minizinc CSP solver. For AMBORO, we are currently using either the Minizinc default solver or the ECLiPse CSP solver. The AMBORO architecture is deliberately structured to make it easy to swap different solvers in and out as the computational and expressive needs of the system change.

AMBORO Architecture and Implementation

The process supported by AMBORO starts with the provision of a problem instance defined in the network and mission models. The mission model describes the tasks involved in the mission, including subtasks and any ordering or other temporal constraints, as well as computational

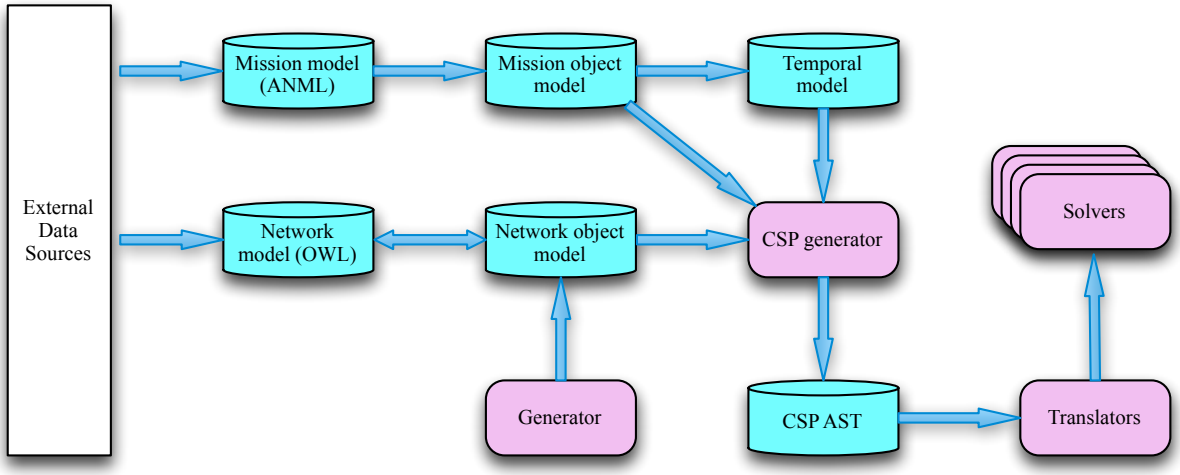

Figure 2: AMBORO Architecture

and communication requirements, which we will refer to as *demands*. The network model describes the computational and communication assets available, including their capacity limits and current configuration. Both models have textual representations in AMBORO, the mission model in the Action Notation Modeling Language (ANML), the network model in the Web Ontology Language (OWL). Both models are then parsed into internal Java data-structures, which are the primary representations operated on within AMBORO. The mission model is then analyzed by a separate module which extracts information about the temporal relationships between different activities, which is used to construct the multi-period schedule representing the progression of mission phases and activities.

The processing module labeled "CSP generator" then uses information from the network model, the mission model, and the temporal model to define an abstract syntax tree represented the desired configuration problem as a constrained optimization problem. This problem is then translated into the appropriate input language for one or more solving engines. At the current time, the solver output is being interpreted by hand. Integration with other Mission-oriented Resilient Clouds (MRC) capabilities will require that the solution format of whatever solver(s) are being employed be mapped back into the entities described in the configuration problem, as represented in the network and mission models.

At this point, we have a fully-functional, end-to-end implementation of AMBORO. Starting from a mission model and a network model, the system will extract the necessary constraints and formulate a multi-period CSP model covering multiple mission phases. This model is then automatically translated into minizinc and submitted to the solver, resulting in a feasible assignment or a notification of infeasibility.

The mission model includes assignable resources, multiple resources required for a given activity, and inter-activity resource constraints (e.g., you must use the same asset for two different activities). The mission model also supports the assignment of subsets of network assets, for example only allowing certain services to run on network nodes with a specified set of capabilities. The network model supports explicit representation of asset reliability, which is carried through the constraint model, all the way to an automatically-generated solution. *Over-provisioning* (the assignment of redundant resources to increase reliability) is currently modeled, but not yet part of the optimization problem.

AMBORO's optimization model has several additional capabilities, including optimizing for minimum distance from a previous assignment to processors (supporting incremental solutions), and a choice of network communication models, encompassing ignoring network liinks, requiring simple connectivity, or feasibility within specified throughput limits on links. Additionally, we are contemplating adding a term to the objective function that would encourage distributing processing demands across processors, as opposed to the natural aggregation that occurs if there is a finite probability of asset failure (because the probability of mission failure rises with the number of unreliable network assets being used).

Network Model

The AMBORO network model is very simple. Shown in Figure 3, the model includes a hierarchical network of *processing elements*, as well as a set of *links* among them. These are the elements that can be allocated as resources, which is all that is required.

There are certainly other aspects to the allocation problem involving the network, such as what demands are allowed on certain links or what services can be allocated to which processing elements. That information is kept in the mission model, which is where the demands are represented. There are additional constraints that represent the "physics"

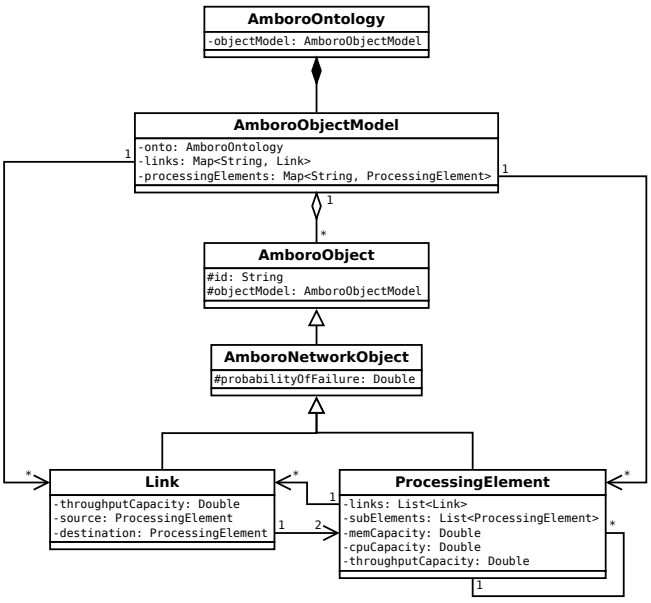

Figure 3: AMBORO Network model

of the network, for example how data flows from one node to another over the available links, or how computational resources used in a sub-element are aggregated into the containing element hierarchy (providing support for virtualization). These constraints are presented in the full constraint model, which space precludes including here.

As previously described, network model instances are stored and exchanged in OWL, which is inter-translated with the in-memory java object model for which Figure 3 provides the class model. The "Generator" function shown in Figure 2 can be used to automatically generate network model instances, which can then be used in the solution process, or back-translated into OWL for export to other tools, or for manual browsing.

Mission Model

Our mission models are represented in the Action Notation Modeling Language (ANML) (Smith, Frank, and Cushing 2008). As a domain modeling language for planning and scheduling applications, ANML provides support for building parameterized task models incorporating temporal constraints such as execution windows and minimum or maximum durations, resource requirements such as necessary tools or equipment, or capacity such as memory usage or network bandwidth. ANML provides a uniform semantics for both precondition/effect models of planning and task decomposition. A further advantage to ANML is that it is specifically designed to facilitate translation into constraint models, indeed has a semantics that is *defined* in terms of relationships represented as constraints. In this section, we present both the ANML representation of mission plans, and the ontology in which mission plans are represented internally.

Tactical Recovery of Aircraft and Personnel (TRAP) Planning Model

Figure 4 shows a simple TRAP mission task model, consisting of two main phases, corresponding to mission planning and execution. Arrows indicate precedence relationships on time points, showing, for example, that three different organizations must be involved in mission planning, all within the mission planning phase. Not shown graphically, but modeled in the ANML mission model presented below, is the requirement that all three of these planning activities happen at the same time. In the second phase, the relationships are different. Air support (AWACS and longer-range air cover provided by fighters) must be in place before close air support (provided by Apaches or other helicopters) can deploy, which must in turn be in place before any ground operations commence. Similarly for the mission return: each level of support must be withdrawn in order.

Additionally, these tasks all come with *resource requirements*. As shown in Figure 5, the planning phase requires local computing, storage, and network capabilities for each organization involved in the planning process, as well as a router that supports communication among them and back to the Global Information Grid (GIG) or other backend, large-scale data storage and processing resource. If any of these resources fail, then the planning phase cannot be completed, at least not until new resources are allocated. In Figure 5, the network has been configured such that there are redundant network paths among all the nodes. At least one of the routers must remain functional in order to maintain the required network connectivity.

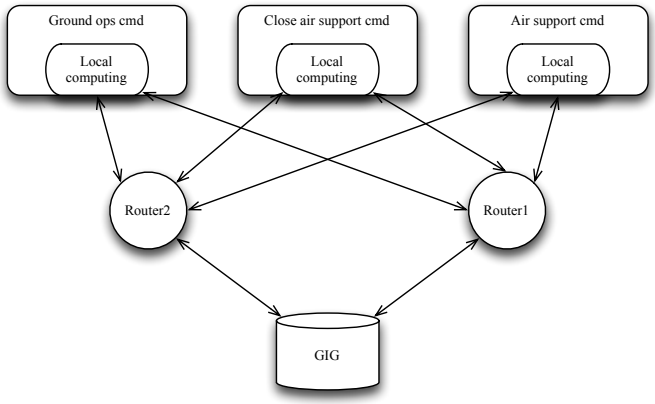

Figure 5: Network resources required for the planning phase.

Figure 6 shows a similar picture of network resources required for the operational phase of the mission. In this picture, all operational communication is routed through the Airborne Warning and Control System (AWACS) aircraft. Those communications include back to the various organizational commands, as well as to the various support aircraft involved. There is no communication link to the retrieval team, the assumption being that they are under some kind of communications restriction. One resource directly related to

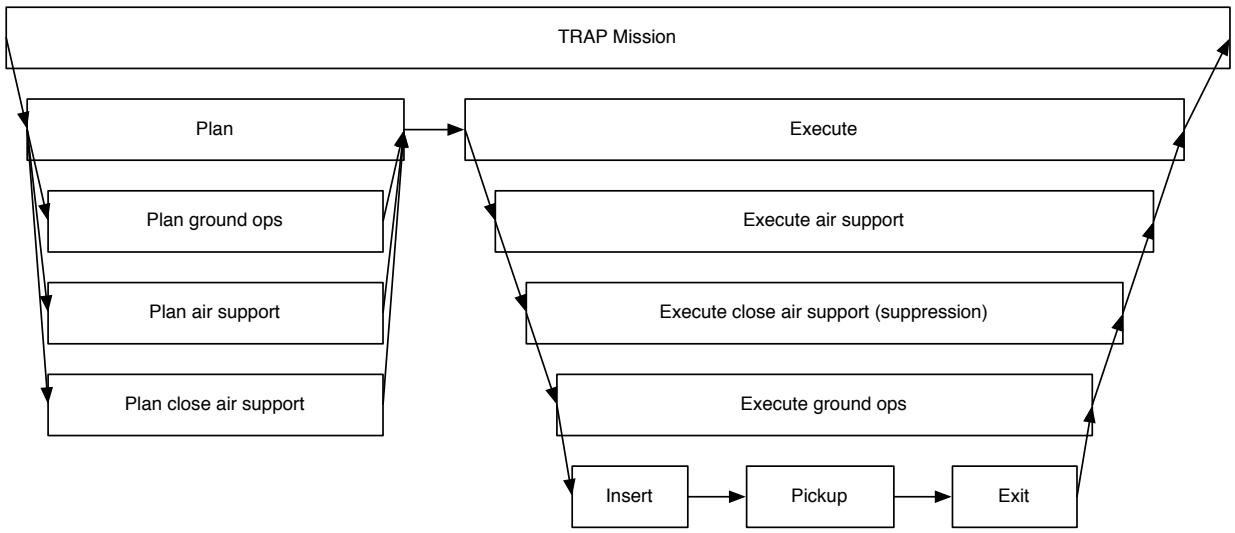

Figure 4: A simple TRAP mission plan

the retrieval team is their on-board computing resources. In particular, they will have pre-loaded map information and probably other data, in preference to have it sent during operational phase. Again, all of these resources must be functional, in order for the mission to succeed. This model can be complicated by adding the possibility of redundancy, by modeling decreased effectiveness rather than outright failure (e.g., do the mission anyway, but with potentially out-of-date imagery), and by modeling Byzantine failures.

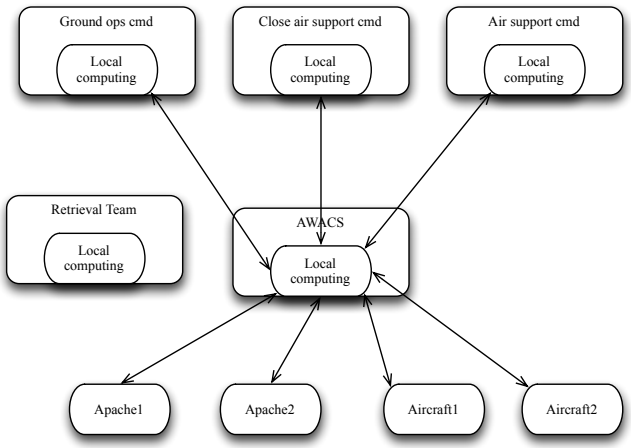

Figure 6: Network resources required for the operational phase.

ANML is a very expressive domain modeling language and could be used to represent a much more complex planning model, including for example alternative task decompositions, more complex constraints on resource assignments, or more general task parameters. The current AMBORO implementation supports more general resource assignments, but does not support alternative task breakdowns.

ANML Mission Planning Model

Here we present an ANML version of the planning model shown in Figure 4. First is the definition of the top-level action:

```
define action TRAP_mission() [duration]
{
  duration <= 360;  // 6-hour time limit
 [all] contains ordered(TRAP_plan(),
                        TRAP_execute());
}
```

This defines the top-level action as taking at most 6 hours, and requiring two subactions, `TRAP_plan` and `TRAP_execute`.

Next we define the planning action:

```
define action TRAP_plan() [duration]
{
  [all] contains {p1: TRAP_plan_ground();
                  p2: TRAP_plan_air();
                  p3: TRAP_plan_close();};

  start(p1) == start(p2) == start(p3);
  end(p1) == end(p2) == end(p3);

  [all] (status(Router1) == OK) |
        (status(Router2) == OK);
}
```

The planning action has three subactions, which are all constrained to start and end at the same times. In addition, it must be the case that either `Router1` or `Router2` is operational over the entire duration of `TRAP_plan`.

Here is one of the planning subactions:

```
define action TRAP_plan_ground()
{
  [all] status(ground_cmd_computing) == OK;
}
```

This very simple action contains a single condition, which is that the computing environment local to ground command is operational. We could also specify starting and ending times or a duration, but need not at this point: this model defines a sufficient set of constraints for a mission plan to be correct, rather than all of the constraints. The other three planning subactions are similar.

Next we define the execution subaction and one of its subactions:

```
define action TRAP_execute() [duration]
{
  [all] contains {e1: TRAP_execute_ground();
                  e2: TRAP_execute_air();
                  e3: TRAP_execute_close();};

  start(e1) <= start(e2) <= start(e3);
  end(e1) >= end(e2) >= end(e3);


  [all] status(AWACS_comms) == OK;
}

define action TRAP_execute_ground()
{
  [all] status(ground_cmd_computing) == OK;
}
```

The example presented here is simplified for clarity and brevity. AMBORO is capable of accepting models that include explicit resource assignments, for example making the decision to use `Router1`, `Router2`, or both an explicit part of the optimization.

Abstract Constraint Model

As described above, the CSP Generator builds an Abstract Syntax Tree (AST) constraint model from information contained in the network and mission models. To the extent practical, this model is abstracted away from the use of specific solver technologies (e.g., Constraint Logic Programming (CLP) versus Mixed-Integer Linear Programming (MILP)) and specific formulations (e.g., linear versus bilinear versus quadratic versus hybrid discrete/math models).

In this section, we present the current state of the abstract constraint model, including recent extensions to represent resource assignment to demands, and probability estimates for the failure of individual network assets. This is a formal specification of the model constructed by the "CSP generator" as shown in Figure 2. We start by defining an instance of a our CSP problem $\mathcal{C}$ as a tuple

$$\mathcal{C} = \langle \mathbf{E}, \mathbf{L}, \mathbf{D} \rangle$$

comprising

- a set $\mathbf{E}$ of *processing elements*,
- a set $\mathbf{L}$ of *links*, and
- a set $\mathbf{D}$ of *demands*.

In this section, we define each of these elements. Subsequent sections will discuss their interaction, and the constraints we add as a result. Constraints to be added to the model are defined in numbered equations.

All constraints in this model are expressed in time-free terms. AMBORO constructs a multi-period model by replicating the static model across periods, with inter-period links as needed for things like allocations to activities that span multiple periods. CPU demand is expressed as a rate requirement (e.g. MIPS). Communications demand is expressed as a requirement for a specified data rate. Memory demand is expressed as an amount of memory that must be allocated, out of a finite store.

Processing Elements

A processing element $e \in \mathbf{E}$ has the following attributes:

- a cpu capacity[3]: $\mathsf{cpu}(e) \mapsto \mathbb{R}^+$
- a memory capacity: $\mathsf{mem}(e) \mapsto \mathbb{R}^+$
- a set of sub-elements[4]: $\mathsf{sub}(e) \mapsto 2^{\mathbf{E}}$

Memory and CPU (and disk, if needed) are all sufficiently similar as currently modeled that, while we define the required attributes for all of these capacity resources, we only present the constraints for CPU. For now, all other capacity constraints local to processing elements look exactly the same as CPU.

A processing element also functions as a node in a network, defined by links, as described below. In addition to the throughput capacities defined on links, we also define a node throughput capacity associated with the processing element e:

$$\mathsf{rate}(e) \mapsto \mathbb{R}^+$$

Finally, a processing element is susceptible to failure. We define the probability that the processing element e will function correctly (i.e., provide the required cpu, memory, and networking capabilities):

$$\mathsf{prob}(e) \mapsto [0.0, 1.0]$$

Links

Communication connectivity between processing elements is provided by *links*. Links are directional, with the following attributes for a link $l \in \mathbf{L}$:

- a source processing element: $\mathsf{src}(l) \mapsto \mathbf{E}$
- a destination processing element: $\mathsf{dest}(l) \mapsto \mathbf{E}$
- throughput capacity: $\mathsf{rate}(l) \mapsto \mathbb{R}^+$

Demands

In this section, we address cpu and communication demands. The model and constraints for memory demands looks just like that for cpu demands. To differentiate between the different types of demands, we define subsets

[3] $\mathbb{R}^+$ denotes the set of non-negative real numbers. Any attribute denoting a value in $\mathbb{R}^+$ must be explicitly constrained to be ≥ 0, unless it's a constant.

[4] The symbol $2^{\mathbf{E}}$ denotes the *power set* of $\mathbf{E}$: the set of all subsets of $\mathbf{E}$, including $\mathbf{E}$ and the empty set $\emptyset$.

of $\mathbf{D}$: D_{cpu} and D_{comm}, each comprising all of the CPU and communication demands, respectively. CPU demands $d \in D_{cpu}$ have the following attributes:

- a demand level: $\mathsf{demand}(d) \mapsto \mathbb{R}^+$
- a set of processing elements to which the demand may be assigned: $\mathsf{allowed}_E(d) \in 2^{\mathbf{E}}$
- a variable [5] $\mathsf{orig}(d)$, which will be assigned a value drawn from $\mathsf{allowed}_E(d)$

A communication demand $d \in \mathbf{D}_{comm}$ has

- a demand level: $\mathsf{demand}(d) \mapsto \mathbb{R}^+$
- a set of processing elements to which the source may be assigned: $\mathsf{allowed}_S(d) \in 2^{\mathbf{E}}$
- a set of processing elements to which the source may be assigned: $\mathsf{allowed}_D(d) \in 2^{\mathbf{E}}$
- a set of *allowed links*[6]: $\mathsf{allowed}_L(d) \mapsto 2^{\mathbf{L}}$
- a variable $\mathsf{src}(d)$, which will be assigned a value drawn from $\mathsf{allowed}_S(d)$
- a variable $\mathsf{dest}(d)$, which will be assigned a value drawn from $\mathsf{allowed}_D(d)$

Processing Element Constraints

Processing elements are defined in a part/whole hierarchy of elements and sub-elements. For processing elements e_i, e_j, where $i \neq j$:

$$e_i \in \mathsf{sub}(e_j) \Rightarrow e_i \notin \mathsf{sub}(e_k), \forall k \neq j$$

and if we define $\prec$ as the transitive closure of the sub-element relation, then

$$e_i \prec e_j \Rightarrow e_j \not\prec e_i$$

These "constraints" are much more likely to be enforced as a property of the network model than to appear explicitly in the CSP to be solved.

There are two possible views of the processing element hierarchy. In the *aggregate* model, processing elements at any level in the hierarchy impose constraints corresponding to usage attributes, interpreted as resource limits to be compared to their aggregate-utilization attributes. For processing elements having no sub-elements, the aggregate-utilization attributes are set or solved for directly (see Subsection). For processing elements with sub-elements the aggregate-utilization attributes are computed from the aggregate-utilization attributes of their sub-elements. We define a variable:

- aggregate CPU utilization: $\mathsf{aggcpu}(e) \mapsto \mathbb{R}^+$

and add constraints:

$$\forall e \in \mathbf{E}, \mathsf{aggcpu}(e) \leq \mathsf{cpu}(e) \tag{1}$$

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggcpu}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{aggcpu}(e') \tag{2}$$

This is a good model for aggregated global resources such as power or comm. bandwidth, where at any level of the hierarchy the sum of the budgets for the next level down may be more than the capacity limit imposed (it is assumed that not everyone will draw their maximum budget at the same time).

In the *budget* model, processing element capacities impose constraints both down the hierarchy (resource limits) and up the hierarchy (resource demands). In this case, we replace the constraint 2 above with

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggcpu}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{cpu}(e') \tag{3}$$

This is more appropriate for something like weight, or power budgets for sub-assemblies that don't get switched on and off. There is no requirement that either the aggregate or budget models be uniformly applied in a given processing element hierarchy; both may be needed, in different places.

CPU Demand Constraints

CPU and memory demands are imposed by constraining the corresponding aggregate utilization. For CPU[7]:

$$\forall e \in \mathbf{E}, \quad \mathsf{aggcpu}(e) = \sum_{d \in D_{cpu}} [\mathsf{orig}(d) = e] \, \mathsf{demand}(d) \tag{4}$$

Note that $\mathsf{sub}(\mathsf{orig}(d))$ must equal $\emptyset$ (i.e., the processing element to which d is applied may not have any sub-elements). This can be implicitly enforced via the membership of $\mathsf{allowed}_E(d)$.

Communication Demand Constraints

We can view the set of processing elements $\mathbf{E}$ and any set of links $L \subseteq \mathbf{L}$ in a given CSP model as a directed graph $G = \langle \mathbf{E}, L \rangle$, with vertices $\mathbf{E}$ and edges L, where each edge $l \in L$ is labeled with $\mathsf{rate}(l)$. Then G fits the definition of a specialized form of directed graph called a *flow network*.[8] Because different communication demands are represented as different flows, with distinct sources and sinks, we need to represent this as a *multi-commodity* flow problem, with each demand $d \in D_{comm}$ corresponding to a different commodity. We define the following with respect to a given set of links $L \subseteq \mathbf{L}$, with $l \in L$, $d \in D_{comm}$ and $e \in \mathbf{E}$:

The source and destination of a communication demand must be distinct:

$$\forall d \in D_{comm}, \quad src(d) \neq dest(d) \tag{5}$$

New variables:

[5] Arguably, variables should be defined in subsequent sections that define the optimization problem. We leave some of them as attributes of the different network entities because, depending on the model, a given attribute may move back and forth between being a variable and being a constant.

[6] The symbol $2^{\mathbf{L}}$ denotes the *power set* of $\mathbf{L}$: the set of all subsets of $\mathbf{L}$, including $\mathbf{L}$ and the empty set $\emptyset$.

[7] The notation used here is the Iversen bracket which is defined by $[S] = \begin{cases} 0 & \text{if } S \text{ is false} \\ 1 & \text{if } S \text{ is true} \end{cases}$

[8] `http://en.wikipedia.org/wiki/Flow_network`

- demand flow on a link: $\mathsf{flow}_L(l,d) \mapsto \mathbb{R}^+$
- demand inflow at a processing element: $\mathsf{inflow}_L(e,d) \mapsto \mathbb{R}^+$
- demand outflow at a processing element: $\mathsf{outflow}_L(e,d) \mapsto \mathbb{R}^+$

Demand inflow and outflow at a processing element is defined by the flow on the connected links:

$$\forall e \in \mathbf{E}, \quad \forall d \in D_{comm}, \quad \mathsf{inflow}_L(e,d) = \sum_{l \in L} [\mathsf{dest}(l) = e]\, \mathsf{flow}_L(l,d) \tag{6}$$

$$\forall e \in \mathbf{E}, \quad \forall d \in D_{comm}, \quad \mathsf{outflow}_L(e,d) = \sum_{l \in L} [\mathsf{src}(l) = e]\, \mathsf{flow}_L(l,d) \tag{7}$$

Demand can only flow on allowed links:

$$\forall l \in L, \forall d \in D_{comm} : l \notin \mathsf{allowed}_L(d), \quad \mathsf{flow}_L(l,d) = 0 \tag{8}$$

The following constraint enforces conservation of flow at each node in the network (each processing element) for each demand:

$$\forall e \in \mathbf{E}, \quad \forall d \in D_{comm} \quad \mathsf{inflow}_L(e,d) + [\mathsf{src}(d) = e]\, \mathsf{demand}(d) = \mathsf{outflow}_L(e,d) + [\mathsf{dest}(d) = e]\, \mathsf{demand}(d) \tag{9}$$

Note that according to this definition, there is no requirement that a given communication flow use a single path from one processing element to another. The throughput required may be spread over any or all of the possible paths between the two processing elements.

Note as well that representing dataflow as a material flow is a potentially-misleading simplification: data may be encrypted, filtered, decoded/expanded, or in other ways made larger or smaller at a given processing element, thus violating the conservation defined in this section and the next one. As long as the change in data size can be mapped to a change in required throughput, this can be added to the current model fairly easily.

Link and Node Rate Constraints

For convenience we define total flow on each link and processing element with respect to a given set of links $L \subseteq \mathbf{L}$, with $l \in L$ and $e \in \mathbf{E}$:

- total flow on a link: $\mathsf{flow}(L)l \mapsto \mathbb{R}^+$
- total inflow at a processing element: $\mathsf{inflow}_L(e) \mapsto \mathbb{R}^+$
- total outflow at a processing element: $\mathsf{outflow}_L(e) \mapsto \mathbb{R}^+$

The following constraints total up the flows across all demands for each link and processing element:

$$\forall l \in L, \quad \mathsf{flow}(L)l = \sum_{d \in D_{comm}} \mathsf{flow}(L)l, d \tag{10}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{inflow}_L(e) = \sum_{d \in D_{comm}} \mathsf{inflow}_L(e,d) \tag{11}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{outflow}_L(e) = \sum_{d \in D_{comm}} \mathsf{outflow}_L(e,d) \tag{12}$$

Now we can specifiy the capacity constraints on links and processing elements:

$$\forall l \in L, \quad \mathsf{flow}(L)l \leq \mathsf{rate}(l) \tag{13}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{inflow}_L(e) \leq \mathsf{rate}(e) \tag{14}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{outflow}_L(e) \leq \mathsf{rate}(e) \tag{15}$$

For now we are assuming that flow for a processing element is constrained independently for in and out flow. But there is an alternative constraint that we could apply if we need to model a device that cannot send and recieve at the same time:

$$\forall e \in \mathbf{E}, \quad \mathsf{inflow}_L(e) + \mathsf{outflow}_L(e) \leq \mathsf{rate}(e) \tag{16}$$

Translating the Abstract Constraint Model

The final step before actually invoking a solver is to translate the problem instance into the appropriate input language. This process is accomplished by walking the Constraint Satisfaction Problem (CSP) AST, emitting the appropriate formulation. In the current implementation, the CSP AST is translated into MiniZinc, which can be used as input for either the default MiniZinc solver, or as input to Eclipse. As a way of showing the kinds of mappings that are required, FIgures 7 and 8 provide examples of the current translation from the formal model into MiniZinc.

There are quite a few more constraints in this model, including those defining how computational resources are aggregated within the element/sub-element hierarchy (i.e., how virtualization is modeled), and the constraints defining communication flows over the network, but this will provide a sense of the kinds of translation required.

This translation needs to be implemented in the inverse direction as well. Output format from many solvers, including MiniZinc, can be heavily tailored, but the variable assignments and costs must be mapped back into the entities in the mission and network models. This requires effectively inverting the two-step translation first from the network and mission models into the CSP AST, and then from there into the solver-specific code shown in this section.

Conclusion and Future Work

AMBORO's uniform, constraint-based representation for modeling mission tasks, cloud resources and the current set of assignments of resources to tasks provides several benefits, including easy extension of the model to reflect new types of missions, tasks, network components, and operational requirements such as QoS guarantees, as well as the presence of a wide variety of implemented solvers. Other advantages to a constraint-based approach include the ability to define partial solutions, culprit identification in the case of infeasibilities or other conflicts, incremental solutions as additional tasks are added, and the use of local search methods.

The work reported here is from the first six months of a planned four-year project. At this point, we have implemented and validated an end-to-end AMBORO system. To date, the network and mission models on which the system has been tested are small, and the probabilities from the which the likelihood of mission success is computed

```
%-----------------------------------------------------------------------------
% Processing Elements

% FORMAL: function cpu(E) -> R+
array [Elements] of float: cpu;
constraint assert(forall (e in Elements) (cpu[e] >= 0.0), "cpu must be >= 0.0");

% FORMAL: function sub(E) -> 2^E
array [Elements] of set of int: sub;

% FORMAL: function aggcpu(E) -> R+
array [Elements] of var float: aggcpu;
constraint forall (e in Elements) (aggcpu[e] >= 0.0);

% FORMAL: function rate(E) -> R+
array [Elements] of float: elementRate;
constraint assert(forall (e in Elements) (elementRate[e] >= 0.0), "rate must be >= 0.0");
```

Figure 7: Adding constraints on processing elements and network links

```
%-----------------------------------------------------------------------------
% Demands

% FORMAL: set D_cpu of CPU demands
int: nCpuDemands;
set of int: CpuDemands = 1..nCpuDemands;

% FORMAL: set D_comm of communication demands
int: nCommDemands;
set of int: CommDemands = 1..nCommDemands;

% FORMAL: function demand(D_cpu) -> R+
array [CpuDemands] of float: cpuDemand;
constraint assert(forall (d in CpuDemands) (cpuDemand[d] >= 0.0), "cpuDemand must be >= 0.0");

% FORMAL: function allowed_E(D_cpu) -> 2^E
array [CpuDemands] of set of int: cpuAllowedElements;

% FORMAL: function orig(D_cpu d) -> allowed_E(d)
% Implemented as an array of zero/one variables.
array [CpuDemands,Elements] of var {0,1}: cpuOrig;
constraint
  forall (d in CpuDemands) (
    % Only one element can be 1 for each demand.  This is the chosen
    % element to fill this demand.
    1 = sum(e in Elements)( cpuOrig[d,e] )
  );
constraint
  forall (d in CpuDemands) (
    forall (e in (Elements diff cpuAllowedElements[d])) (
      % Exclude elements that are not allowed to be used for this demand
      cpuOrig[d,e] = 0
    )
  );

constraint
  forall (d in CpuDemands) (
    forall (e in (Elements diff commAllowedSrcs[d])) (
      % Exclude elements that are not allowed to be src for this demand
      commSrc[d,e] = 0
    )
  );
```

Figure 8: Adding demand constraints

are largely independent. We are currently in the process of adding complexity and scale in all three areas: larger networks, more complex mission models, and more complex probability computations. Further along in the project, we will also be extending from the simple failure probabilities described here to a more complex (but not yet defined) notion of "compromised" assets, where different kinds of compromise have different, possibly stochastic effects on mission effectiveness.

All of these extensions pose potential computational challenges. The question of network scale we have already addressed: our previous work on CARINAE demonstrated an ability to scale to networks of up to a million nodes, with only limited optimizations in our formulation. Added complexity in the mission model may affect solving time in any of several ways. Adding additional mission phases or additional levels of sub-tasks to the current multi-period model increases the size of the problem linearly. Given the relatively weak coupling of adjacent periods, this should not have a great effect on computational effort. Additional inter-task constraints, unordered subtasks, or permitting alternative task decompositions are the extensions most likely to add difficulty to scaling up.

Moving to a more general and more complex stochastic model has the potential to add significant difficulties. The overall MRC program is currently in an early stage, thus we do not know exactly what additional complexities will be required. Depending on specific details for these extensions, our approach may range from off-line model simplification and exploitation of what independencies exist, using an existing probabilisitic modeling language such as Figaro (Pfeffer 2009), to numeric approximations exploiting the fact that prior failure probabilities tend to be very close to zero, to hybrid optimization methods involving partitioning a large, non-convex solution space, as for example in (Lamba et al. 2003).

References

Haigh, J.; Harp, S. A.; and Payne, C. N. 2010. Aimfirst: Planning for mission assurance. In *Proceedings of 5th International Conference on Information Warfare and Security*.

Hyser, C.; McKee, B.; Gardner, R.; and B.J.Watson. 2007. Autonomic virtual machine placement in the data center. Technical Report HPL-2007-189, HP Labs. `http://www.hpl.hp.com/techreports/2007/HPL-2007-189.pdf`.

Lamba, N.; Dietz, M.; Johnson, D. P.; and Boddy, M. 2003. A method for global optimization of large systems of quadratic constraints. In *2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction*.

Michalowski, M.; Boddy, M.; and Carpenter, T. 2010. Coordinated management of large-scale networks using constraint satisfaction. In *Working Notes of the 2010 AAAI Workshop on Intelligent Security (SecArt)*.

Padal, P.; Shin, K. G.; Zhu, X.; Uysal, M.; Wang, Z.; Singhal, S.; Merchant, A.; and Salem, K. 2007. Adaptive control of virtualized resources in utility computing environments. In *EuroSys*.

Pfeffer, A. 2009. Figaro: An object-oriented probabilistic programming language. Technical report, Charles River Analytics.

Sarmenta, L. 2002. Sabotage-tolerance mechanisms for volunteer computing systems. In *CCGrid*.

Singh, A.; Korupolu, M.; and Mohapatra, D. 2008. Server-storage virtualization: Integration and load balancing in data centers. In *Proceedings of the ACM/IEEE Conference on Supercomputing*.

Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. In *International Conference on Automated Planning and Scheduling*.

Sonnek, J.; Chandra, A.; and Weissman, J. 2007. Adaptive reputation-based scheduling on unreliable distributed infrastructures. In *TPDS*.

Wang, Y., and Wang, X. 2010. Power optimization with performance assurance for multi-tier applications in virtualized data centers. In *IEEE Online Conference on Green Computing*.

Wood, T.; Shenoy, P.; Venkataramani, A.; and Yousif, M. 2007. Black-box and gray-box strategies for virtual machine migration. In *In Proceedings of NSDI*.