Automated Redeployment of Real-Time Systems Informed by User-Provided Workflows

James Edmondson, Aniruddha Gokhale, Sandeep Neema
Dept of EECS, Vanderbilt University
Nashville, TN 37212, USA
{*james.r.edmondson,a.gokhale,sandeep.neema*}*@vanderbilt.edu*

***Abstract*—Distributed, real-time and embedded (DRE) systems are a subset of computing applications that require stringent quality-of-service despite operating in environments with scarce CPU, memory, or network resources. A general lack of automated tools for redeploying DRE systems in a timely manner upon incurring failures or degradation in QoS have required an engineer or developer to reboot the system and redeploy the important components manually. Whether this type of delay is acceptable to users of the DRE system depends on the context of the application usage, but for mission-critical applications, such outages and delays can result in loss of money and lives. In this paper, we discuss our efforts and elaborate on the challenges in the area of middleware to faciliate dynamic redeployment of continuously available and potentially mission-critical DRE systems.**

***Keywords*-redeployment, constraint satisfaction, optimization, quality of service, cyberphysical systems**

I. Introduction

One of the major difficulties with pervasive distributed, real-time and embedded systems (DRE) is that they are continuous systems that require solutions to active problems within time constraints that cannot be determined offline because timing is absolutely critical—often at the expense of money or lives. It is within this context that we are developing middleware solutions into the Multi-Agent Distributed Adaptive Resource Allocation (MADARA) framework, an open source toolkit for DRE system developers. Our current work in progress involves the timely redeployment of distributed applications optimized in accordance to user-defined constraints and workflows.

To concretely describe our work, consider a motivating scenario shown in Figure 1. The context of Figure 1 could be a disaster recovery scenario, *e.g.*, an earthquake-ravaged metropolitan area where thousands of ground-based drones have been deployed to search for survivors and gas leaks. Because of the destruction, controllers of the drones are restricted to satellite connections and the bandwidth available over this limited network resource is only enough for a handful of dedicated sessions. Consequently, the controllers can only maintain communication with a limited subset of the drones.

An application workflow is specified which allows these special drones to serve as collection points for the important sensor readings, images, and audio from the disaster zone as well as the communication point to human-based controllers. As the drones move around the area, an earlier optimal deployment of these special drones may become out-of-date, and a redeployment of the specialized collection and controller communication logic may be necessary in order for the special drones to be in range of their group within the workflow. The drones are also battery-powered so any unnecessary computation or communication should be minimized to reserve power.

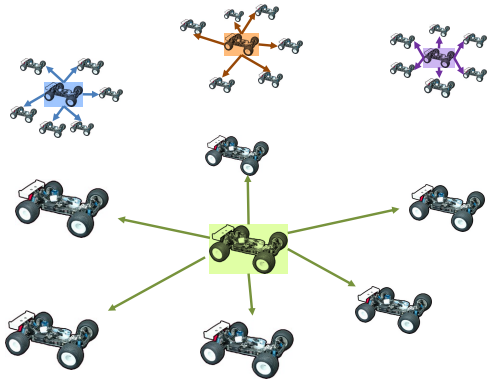

Figure 1. Example Deployment Workflow of Four Drones Broadcasting in and Collecting from Dynamic Groupings

Consequently, this DRE scenario imposes the following requirements for a (re)deployment solution:

1) Users must be able to define a flexible deployment workflow for thousands of drones
2) Algorithms for approximating the workflow against the current network conditions must be able to execute quickly (sub-second runtime is preferred). Failure to do so may result in loss of drones or inability to find survivors.
3) The middleware should also be able to detect failure or success of a running application as this may effect the deployment. In fact, if a program fails (*e.g.*, a radioactivity detecting sensor fails or reports dangerous levels), the deployment may need to change so that the

drone groups keep their distance from contaminated sectors, as losing a drone would result in potentially finding less survivors and shorter system uptime.

4) In addition to being able to specify a workflow, DRE system integrators may also want to specify constraints on or between nodes. For instance, the special collecting drones may have a deployment constraint that specifies that the drone must have more than 20% battery power remaining or else the deployment must change immediately (unless all other drones are operating at 20% or less power).
5) This motivating scenario does not mirror an area coverage problem and actually reflects an optimization problem on top of the subgraph isomorphic problem (an NP complete problem) where subgraphs may not communicate directly at all. The subgraphs are also arbitrarily defined by the user according to the requirements of their distributed applications, and we consider the "optimal deployment" to be the one that minimizes total latency along the edges of all subgraphs.

In this paper, we discuss ongoing work in this problem in applying a heuristic called the Comparison-based Iteration by Degree (CID) heuristic to approximate optimal redeployments, and augmenting deployment middleware to inform the participants of the deployment (the drones) that a redeployment is necessary. The remainder of this paper is organized as follows. In Section II, we look at related work in deployment frameworks and constraint satisfaction problem solving. In Section III, we discuss our solution approach involving knowledge and reasoning, advanced AI and heuristics, and deployment middleware. Section IV highlights ongoing challenges in our work that are actively being researched and addressed. Finally, Section V discusses the contributions of the paper as well as future work.

II. Related Work

A. Deployment and Testing

Most networked deployment and testing infrastructures [1], [2] use centralized controllers to instrument, control, and sequence deployment entities. None of them appear to support real-time redeployment and these centralized controllers are a bottleneck and central point-of-failure that may result in loss of connectivity and control of the real-time system until it is rebooted. This is an unacceptable risk for DRE systems.

B. Constraint Satisfaction Problem Solving

The drone deployment problem requires approximations of a constraint satisfaction problem involving thousands of nodes within a timely manner. Constraint satisfaction solvers [3], [4] have managed to scale approximations and optimal solutions to hundreds and even thousands of features and nodes. Unfortunately, these solutions take time (generally hundreds to thousands of seconds—afterall, CSPs are NP complete problems).

Other research has gone into local searches like genetic algorithms [5] and combinations of neural networks [6] or knowledge and reasoning [7] to help the local searches converge to optimal solutions quickly. However, these solutions still take hundreds of seconds and may even take longer than some of the CSP solutions noted earlier.

C. Area Coverage and Sensor Networks

Sensor networks are a widely studied technique for building routing networks for information in a resource-constrained environment. In general, these sensor networks target deployments of a single application topology in such a way that routing is accomplished across an entire network (solving the Area Coverage problem) via potential fields [8], quorums [9], and other techniques. The main difference between these sensor network solutions and ours is that we're targeting a different type of deployment problem. Our scenario does not need full area coverage, which is not NP complete and easier to solve. The motivating scenario requires a drone cloud that is separated by controller according to an arbitrary data flow that may not allow or encourage communication between disparate subgraphs. It's important to stress that each subgraph could be hierarchical or cyclic structure and Figure 1 is meant as a simplified example, but it's also not an area coverage problem. Further, the subgraphs may involve secondary functions–like sensing radiation versus thermal imaging–that are orthogonal to the function of searching for survivors.

III. Solution Approach

In this section we outline our approach that is integrated into the Multi-Agent Distributed Adaptive Resource Allocation (MADARA) suite of C++ tools to tackle the problems of redeployment in online DRE systems. Section III-A highlights our progress in heuristics and genetic algorithms which can approximate optimal application deployments in milliseconds. In Section III-B, we describe our distributed knowledge and reasoning system which allows deployments to be approached from a host-agnostic, knowledge-centric perspective. In Section III-C, we outline our static deployment tools for one shot initial configurations of processes. We plan to integrate the heuristics into our existing middleware to facilitate real-time redeployments in continuous systems.

A. Approximating Optimal Workflows

Recently, the MADARA suite has adopted the Comparison-based Iteration By Degree (CID) heuristic [10] for approximating networks of processes according to a user-provided workflow. This tool is the basis of our efforts in approximating a distributed application, and we have further optimized this heuristic by reducing its runtime

Table I
WORKFLOW DESCRIPTION FOR FOUR SPECIAL DRONES, EACH COLLECTING FROM A QUARTER OF THE DRONE POPULATION

12 0	→	[size/4)
size/4	→	[size/4, size/2)
size/2	→	[size/2, 3*size/4)
3*size/4	→	[3*size/4, size)

footprint from seconds to tens of milliseconds for tens of thousands of processes, compared to hundreds of seconds to hours with traditional CSP solvers and local search techniques.

The CID heuristic works by analyzing a deployment (an example of which is shown in Table I and optimizing the network according to the degree (essentially, the connectivity) of nodes in the associated graph. The higher the degree of the node in the graph, the more stress that a bad deployment choice affects the rest of the deployment because the user has indicated that this node is important and is communicating with others. For our motivating scenario, each node is a drone in the deployment.

Algorithm 1 CID Heuristic

```
1:  for all i ∈ deployment_graph do
2:    if degree (i) > 0 then
3:      solution[i] ← best_candidate (utilities[degree(i)])
4:    end if
5:  end for
6:  for all i ∈ deployment_graph do
7:    if degree (i) > 0 then
8:      for j ∈ connections(deployment_graph, i) ∧ j ∉
        solved(solution) do
9:        solution[j] ← best_candidate (latencies[i])
10:     end for
11:   end if
12: end for
13: for all i ∈ deployment_graph ∧ i ∉ solved(solution) do
14:   solution[i] ← best_candidate (utilities[size])
15: end for
```

The CID Heuristic shown in Algorithm 1 begins by iterating over the deployment and placing candidates based on lowest latency for the degree. We place our lowest total latency candidates on the nodes with the highest connectivity (lines 1-5) and then iteratively fill in their closest neighbors when possible on lines 6-12 (*i.e.*, when it does not conflict with other high degreed nodes in the DRE application workflow). The final phase of the CID (lines 13-15) deals with nodes that are not connected to the rest of the DRE application workflow, *e.g.*, for worker drones that do not communicate with the drone collector and serve as sentries, data analyzers, or passive entities whose results can be processed or collected offline (non-mission critical).

The algorithm produces optimal solutions for deployments with disjoint subgraphs, as is the case with our motivating scenario, and does so in less than 20 milliseconds for deployments of 10,000 drones or components using modern PC architectures. For drone hardware operating in the mhz range, this may take up to a second, but the number of supported drones is unlikely to be so high if we're doing computation on a drone processor. It does not produce optimal deployment in all cases because that would require a solution that solves the subgraph isomorphic problem, a known NP Complete problem [11].

A large portion of our current investigations are specialized in finding out what deployments result in poorer performance and the modifications we can make to correct this (*e.g.*, changes to the heuristic or pairing the heuristic with local search techniques or backtracking and randomization). The goal of producing our heuristics is to not only to solve the motivating scenario but redeployment of DRE systems in the general case.

B. Informing Processes of Change

The MADARA suite of tools includes a distributed Knowledge and Reasoning Language (KaRL) engine that facilitates a host-agnostic method of data processing that is ideal for informing the drones that a redeployment is necessary. This engine was developed for real-time, continuous systems and is capable of processing knowledge rules in mhz on ghz processors and khz on Atom-based drone processors. It also enforces temporal, priority, and update consistencies for knowledge that is disseminated across the network, which makes it ideal for dealing with faults, failures, and unreliable networks.

C. Deployment Configuration

The MADARA suite includes a flexible deployment and testing suite of tools called the KaRL-Automated Testing Suite (KATS) which provides decentralized, portable process management to DRE developers. KATS presents a simple XML-based interface for specifying executables, command lines, kill times and signals, input and output redirection, and various other operating system-neutral configuration settings for each process in the deployment. This allows DRE system developers in our motivating scenario to define the actions required during a deployment and redeployment, the logic that a collector drone or a sensing drone must execute, and the order in which drone services must launch, including timing delays or dependencies with other drones in the deployment.

Embedded into the KATS tool set is a fine-grained process life cycle that allows for dynamic sequencing with microsecond precision (dependent on network transports available). The lifecycle allows for the drones to barrier on subgraphs to become available or establish preconditions specific to their subgraph.

IV. Remaining Challenges

A. Custom Constraints

Within MADARA and the CID heuristic, we currently have no way to specify custom constraints which may be very important to the appropriate redeployment solution for a DRE system. For instance, if the battery power of a drone is too low to allow it to serve as a special collection drone, then it should not be eligible for a node specified in the deployment as having a high degree.

Once we have a system in place, we will have to incorporate constraint checking mechanisms into Algorithm 1 and local searches in a way that is both robust and efficient. Additionally, we are working on mechanisms to incorporate constraints and workflows into the KATS XML-based input for processes.

B. Better Approximation

The CID heuristic does not necessarily always find an optimal solution (the lowest total aggregated latency across all edges of all subgraphs). The problem we are working on is actually a type of subgraph isomorphism problem—defined as a computational task in which two graphs are given as input and one must determine if one of the graphs is a subgraph (in this case the user-defined workflow is the subgraph) [11].

We are currently working on genetic algorithms and time-limited backtracking to take over the work of approximating the user-provided workflow after being seeded with the results of the CID heuristic and variations of that heuristic. We've had some success with this approach, but more work is needed with local search techniques like genetic algorithms to produce better approximations faster.

C. Addressing Continuous Semantics

Inside of our deployment framework, we need to add mechanisms to address redeployment issues. For instance, what if the process that is being redeployed had side effects—*e.g.*, a database? How do we provide developers with tools to fine-tune how the redeployment behaves?

We are working on enhancements to the KaRL engine to support file transfer to meet the challenge of side effects, and we are investigating extending the process lifecycle of KATS to support new phases like on_success, on_failure, and on_redeploy to allow for changes in deployment logic, constraints, and workflow information into the XML-based input files to provide a more robust solution.

V. Conclusions

In this paper we have outlined the need for tools and techniques to facilitate dynamic redeployments of failing or out-of-date DRE applications. Our ongoing work with tailoring the MADARA suite (open source and downloadable from madara.googlecode.com) seeks to address this need via effective knowledge and reasoning, heuristics, and deployment technologies targeted at continuous, mission-critical DRE systems. Our approach is generic enough to address most types of DRE applications, and fast enough to provide optimized solutions in sub-second time. Though we have made progress toward our goals and pushed the state-of-the-art in this regard, we still have ongoing work in providing custom constraints, better approximations, and mechanisms to address continuous semantics.

References

[1] C. Dumitrescu, I. Raicu, M. Ripeanu, and I. Foster, "Diperf: An automated distributed performance testing framework," in *in 5th International Workshop in Grid Computing*. IEEE Computer Society, 2004, pp. 289–296.

[2] T. Li and T. Bollinger, "T.: Distributed and parallel data mining on the grid," in *Proc. 7th Workshop Parallel Systems and Algorithms*, 2003.

[3] P.-E. Hladik, H. Cambazard, A.-M. Daplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *Journal of Systems and Software*, vol. 81, no. 1, pp. 132–149, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121207000672

[4] J. White, D. Schmidt, D. Benavides, P. Trinidad, and A. Ruiz-Cortes, "Automated diagnosis of product-line configuration errors in feature models," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, sept. 2008, pp. 225–234.

[5] L. Ingber and B. Rosen, "Genetic algorithms and very fast simulated reannealing: A comparison," *Mathematical and Computer Modelling*, vol. 16, no. 11, pp. 87 – 100, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089571779290108W

[6] A. Javadi, R. Farmani, and T. Tan, "A hybrid intelligent genetic algorithm," *Advanced Engineering Informatics*, vol. 19, no. 4, pp. 255 – 262, 2005.

[7] Y. Hu and S. Yang, "A knowledge based genetic algorithm for path planning of a mobile robot," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, april-1 may 2004, pp. 4350–4355.

[8] A. Howard, M. J. Mataric, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," 2002, pp. 299–308.

[9] G. Wang, G. Cao, T. L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *In Proc. of IEEE INFOCOM*, 2005, pp. 2302–2312.

[10] J. Edmondson and D. Schmidt, "Multi-agent distributed adaptive resource allocation (madara)," *International Journal of Communication Networks and Distributed Systems, Special Issue on: Grid Computing*, vol. 5, no. 3, pp. 229–245, 2010.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.