Approximation Techniques for Maintaining Real-time Deployments Informed by User-provided Dataflows Within a Cloud

James Edmondson
Carnegie Mellon Software Engineering Institute
Pittsburgh, PA 15213, USA
jredmondson@sei.cmu.edu

Aniruddha Gokhale, Douglas Schmidt
Dept of EECS, Vanderbilt University
Nashville, TN 37212, USA
{a.gokhale,d.schmidt}@vanderbilt.edu

***Abstract*—Distributed applications are increasingly developed by composing many participants, such as services, components, and objects. When deploying distributed applications into a mobile ad hoc cloud, the locality of application participants that communicate with each other can affect latency, power/battery usage, throughput, and whether or not a cloud provider can meet service-level agreements (SLA). Optimization of important communication links within a distributed application is particularly important when dealing with mission-critical applications deployed in a distributed real-time and embedded (DRE) scenario, where violation of SLAs may result in loss of property, cyber infrastructure, or lives.**

To complicate the optimization process, the underlying cloud environment can change during operation and an optimal deployment of the distributed application may degrade over time due to hardware failures, overloaded hosts, and other issues that are beyond the control of distributed application developers. To optimize performance of distributed applications in dynamic environments, therefore, the deployment of participants may need adapting and revising according to the requirements of application developers and the resources available in the underlying cloud environment.

This paper present two contributions to the study of dynamic optimizations of user-provided deployments within a cloud. First, we present a dataflow description language that allows developers to designate key communication paths between participants within their distributed applications. Second, we describe heuristics that use this dataflow representation to identify optimal configurations for initial deployments and/or subsequent redeployments within a cloud. An experiment is presented to validate the heuristic approaches.

***Index Terms*—heuristics; genetic algorithms; clouds optimization; real time; constraint problems**

I. Introduction

Enterprise distributed real-time and embedded (DRE) systems are mission-critical applications that run in networked processes across heterogeneous architectures under stringent timing requirements and scarce resources [2]. Though enterprise DRE systems were originally associated with avionics, manufacturing, and defense applications, they increasingly focus on a broader class of distributed applications where the right answer delivered too late becomes the wrong answer. Information in DRE systems must therefore be delivered according to stringent quality-of-service (QoS) needs, despite failures and resource limitations [13], [15], [12].

Unlike some enterprise cloud-based applications that deal with service-level agreement (SLA) violations with small surcharges to the cloud infrastructure provider, mission-critical enterprise DRE systems cannot tolerate unresponsiveness. Recurring poor performance may thus result in financial loss and even deaths. DRE systems often require continuous human vigilance to maintain appropriate end-to-end QoS. Moreover, cloud environments do not optimize distributed deployments according to user-defined application dataflows between important participants. To enable next-generation cloud environments to support DRE applications, therefore, they need the following capabilities:

1) A means to specify key communication paths within a distributed application to inform the underlying cloud of what participant interactions should be optimized.
2) Heuristics for optimizing distributed application participant pathways that are identified as important by the user or a monitoring system that informs the cloud infrastructure of heavily utilized pathways.

This paper describes extensions to the *Multi-Agent Distributed Adaptive Resource Allocation* (MADARA) [5], [6] open-source multi-agent middleware, which provides adaptive deployment tools to support next-generation cloud computing capabilities for DRE systems. We have enhanced MADARA to provide a dataflow description language that allows developers to designate key communication paths between participants within their distributed applications. MADARA now also provides genetic algorithms and heuristics that use this dataflow representation to identify optimal configurations for initial deployments and/or subsequent redeployments within a cloud using real-time latency information. In addition, MADARA now provides developers with methods for aggregating latency information via summations of latencies along important paths in the dataflow, which is useful for other approximation techniques that require similar aggregations of latency.

The remainder of this paper is organized as follows: Sec-

James Edmondson completed work while at Vanderbilt University

tion II presents a search-and-rescue scenrio that motivates the need for the specification and heuristics added to MADARA; Section III describes the dataflow description specification and heuristics MADARA uses to minimize end-to-end latency in a DRE application dataflow within a cloud; Section IV analyzes the results of experiments that evaluate how well the MADARA guided genetic algorithms and heuristics approximate a user workflow; Section V compares MADARA with related work on approximation techniques; and Section VI presents concluding remarks.

II. Motivation Scenario

To motivate the need for MADARA, this section presents a scenario that occurs during a search-and-rescue mission where multiple government agencies utilize a cloud of remote-controllable drones within a disaster area. Figure 1 shows this disaster recovery scenario, where remote-controllable drones have been deployed to search for survivors in an earthquake-ravaged metropolitan area. The application dataflow shown

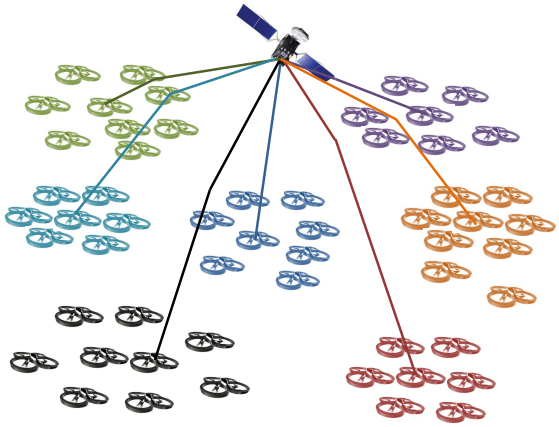

Fig. 1. Motivating Search-and-Rescue Application Scenario

in this figure shows segregated groups of remote-controlled drones in the search-and-rescue mission communicating via satellite with human controllers. Due to the destruction, human controllers of the drones are restricted to satellite connections and the bandwidth available over this limited network resource is sufficient for only a handful of dedicated sessions between humans and the drones searching for signs of life.

Human controllers can thus only maintain communication with a small subset of the drones, called *collector* drones. Each drone has onboard sensors that may allow it to detect radiation, record video, observe and report atmospheric anomalies, detect thermal signatures, and other useful functions, and regardless of which government agency leased time on the drone cloud, each of the automated participants aids in searching for survivors. Each drone is also equipped with a wireless access point that allows it to form/join ad hoc networks and transmit sensor readings, images, or other data.

With all secondary functions (*e.g.*, radioactivity detection) turned on, the drones will quickly run out of power. Moreover, the faster the drones power down, the less survivors that will be found and useful work accomplished. Data communication is a particularly expensive operation that quickly drains batteries. The closer two drones are to each other, however, the lower the latency and the less data resends required across the communication, which extends battery life.

An example application dataflow is shown in Figure 2. The data shown in this figure enables two radiation detectors and has two collector drones communicating with human controllers via satellite links. Each edge in an application dataflow shown in this figure is equally important, *i.e.*, no edge is more important than any other edge that is defined in the application dataflow. The absence of an edge means that the datapath is unimportant or at least that no attempt should be made to minimize latency along that datapath. We also assume that each edge is utilized equally (*i.e.*, no edge is monopolizing traffic unevenly), and this caveat helps us to simplify the optimization problem: *the best deployment will be the one that has the lowest total latency summed across all edges*.

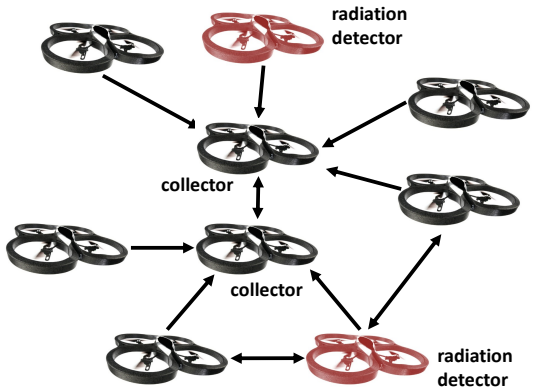

Fig. 2. Application Dataflow Example

As drones move around the area, the optimal deployment of DRE applications that are running on the drones may become outdated. A redeployment of the DRE application may therefore be necessary to ensure these collector drones are in range of their group within the dataflow. This redeployment time is pure overhead and the associated computation time competes with the CPU and memory resources that the human controller needs to view important data, as well as draining precious battery life. If the drones remain computation-bound for too long and lock out their controllers from viewing information or issuing commands, survivors may be missed, drones may crash into buildings or other obstacles, and lives and resources may be lost. For these reasons, the time required for calculating the redeployment should be minimized—preferably a handful of seconds or less.

This motivating scenario consequently requires three main things:

1) An effective way to describe the application dataflow

TABLE I
DATAFLOW DESCRIPTION FOR MOTIVATING DRE APPLICATION

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

2) Algorithms to efficiently calculate the optimal secondary activity according to an arbitrary application dataflow that may not reflect the Area Coverage Problem [1], [14], [7], which is commonly solved with sensor networks.
3) A noticable improvement in network latency along the edges of the application dataflow after using one of the algorithms

III. APPROXIMATION TECHNIQUES IN MADARA

This section describes genetic algorithms and heuristics provided by MADARA to approximate an optimal enterprise DRE application deployment under different constraints. We developed multiple solutions due to memory limitations imposed by different contexts where the solutions are deployed. These solutions are complementary and can be chained together to produce seeds and candidates for other genetic algorithms or heuristics. These heuristics can also be run on all hosts in the cloud or on specific hosts, such as collector drones or a master host.[1]

A. Defining the Dataflow and Identifying Degrees

MADARA optimizes DRE application dataflows from a graph perspective. In particular, it encodes a user-defined deployment dataflow into a graph and use degree information to inform our approximation process. The degree of a node in a graph is the number of connections incident on the node, *i.e.*, it is essentially a connectivity metric. This concept of degree is derived from graph algorithms, as well as distributed and parallel computing.

The degree of a graph is relevant to MADARA because it seeks solutions that minimize the latency or improve the overall utility of the connections between nodes in a DRE application dataflow. The node with the highest degree has the most impact on this overall metric. It is therefore often a major bottleneck in DRE applications.

To show how a degree is imparted from an application dataflow, Table I depicts an actual dataflow description file for our motivating application in Section II, which consists four collector drones, each gathering messages from a quarter of the drone population. The MADARA dataflow description language provides a mapping of directed edges and is ideal for specifying large ranges of values, which maps well to cloud environments. The simplest dataflow description involves a source mapped to a range of destination participants, which are processing elements capable of executing a component or service of a distributed application.

[1]This paper does not specify how cloud hosts agree on a redeployment and assume a distributed voting protocol is used to determine redeployment thresholds (which is how we implement redeployment agreement in MADARA.)

For instance, participant 0 in the first line of Table I has important edges from itself to participant 0 to size / 4, where '[' denotes inclusiveness and ')' denotes non-inclusiveness. Instead of a single participant id, the source participant in the dataflow description language can be a range of IDs, *e.g.*, [0, size/4] → [0, size/4) indicates that important edges exist between each participant in one-fourth of the available participants in the cloud. The number of participants available per host can be potentially infinite, but for DRE systems it should ideally map to the number of processors available or less if threads of execution should be available for certain system threads at all times.

From the dataflow description in Table I, we can make the following observations. There are four special drones, and each are servicing a large portion of the underlying drone network. If the size is set to 12, the logical drone 0 is servicing drones 0-2. Drone 3 services 3-6, drone 6 takes care of 6-8, and drone 9 handles information to and from 9-11. This MADARA deployment specification interface addresses requirement 1 of the motivating DRE application in Section I by providing users a flexible mechanism for specifying a deployment dataflow in a DRE system.

Figure 3 visualizes what a degree in a graph is by labeling the high degree nodes in a user-provided DRE application dataflow. The node with a degree of seven has seven directional

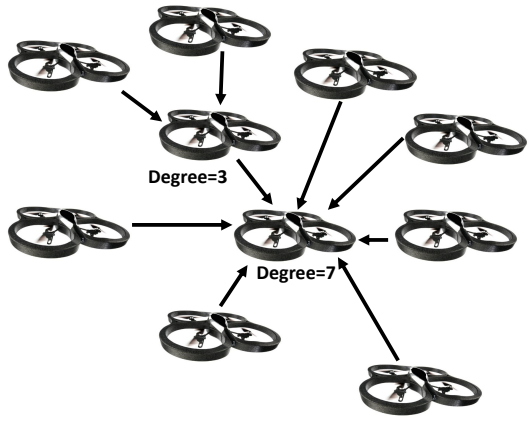

Fig. 3. Degrees in a User-provided DRE Application Dataflow

edges coming in or out of the node. A degree with three signifies that the node has a connectivity of three. Though this paper focuses on using degree information for the motivating DRE application in Section I, our solution techniques are relevant to approximating component placement, optimal resource monitoring, routing, and other problems involving connected graphs.

B. Degree-based Heuristics in MADARA

Two heuristics are discussed below, each targeting a different context of the motivating DRE application. The *Comparison-based Iteration by Degree* (CID) Heuristic (shown in Algorithm 1) is useful for seeding genetic algorithms when the drone has enough memory to hold latency

information of all other drones ($O(N^2)$ space requirement), which can become hundreds of megabytes when thousands of drones or processes are involved.

Algorithm 1 CID Heuristic

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

Algorithm 1 shows how the CID Heuristic begins by iterating over the deployment and placing candidates based on lowest latency available in the cloud for the degree.

The latencies list is a sorted list of latencies between all participants. Thus, latencies[node] is the list of latencies involving a certain node. 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 dataflow).

The final phase of the CID heuristic (lines 13-15) deals with nodes that are not connected to the rest of the DRE application dataflow. For example, this phase could be used 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).

A variant of the CID heuristic we developed called the *Blind CID* heuristic is shown in Algorithm 2.

The Blind CID heuristic is useful for deployments where drones do not have as much memory (O(N) space instead of $O(N^2)$). The drawback is that the Blind CID heuristic is a less informed approximation of the solution than the CID Heuristic and may not find the optimal deployment, which results in less battery life, longer latencies, and more resends of important information.

A key difference between the CID heuristic and the Blind CID heuristic (Algorithm 2) is that the CID heuristic uses the fine-grained latency information from all drones in the network. In contrast, the Blind CID heuristic only uses aggregation of this knowledge in a preparation phase. The Blind

Algorithm 2 Blind CID

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

CID heuristic does use deployment information in the dataflow to prioritize which node of the dataflow to approximate next. It always selects from the best total latency value (essentially the aggregate of a full broadcast from the node), however, rather than the aggregate of best latencies from this node for the degree.

The benefit of the Blind CID heuristic is that the drones need not send their individual latency values to other drones that must make redeployment decisions (O(N) total message complexity unlike the other algorithms). Each node using Algorithm 2 alone has a message complexity of O(1), a message containing an aggregrate latency value for a full broadcast from the node. Sending fewer messages increases battery life for all participants in the dataflow.

C. Genetic Algorithms in MADARA

Not all DRE application dataflows can be solved optimally by the heuristics described in Section III-B. The CID and BCID heuristics are tailored to solve certain types of dataflows like acyclic collector drones and not more complex dataflows like hierarchical or cyclic dataflows. For more complex dataflows, a randomized search technique may be more appropriate.

To complement the heuristics discussed in Section III-B, we therefore developed two genetic algorithms to hone the approximated solution before deciding if a redeployment is necessary for the special drones. Only one of these genetic algorithms—Guided GA shown in Algorithm 3—is guided with degree information.

The Blind GA Algorithm does not use degree information to mutate solutions and instead uses pure randomness when selecting solution chromosomes to mutate.

Before describing the Guided GA Algorithm(see Algorithm 3) and Blind GA Algorithm solutions we briefly describe what constitutes a mutable chromosome in the deployment. Each of these algorithms considers a chromosome as a mapped

Algorithm 3 Guided GA

```
1: mutations ← min + rand() % (max - min)
2: orig_utility ← utility(new)
3: for i → mutations do
4:    new ← solution
5:    if rand() % 5 < 4 then
6:       c1 ← random_degreed_node (dataflow)
7:       c2 ← location(new[good_candidate(utilities)])
8:       while c1 ≡ c2 do
9:          c2 ← location(new[good_candidate(utilities)])
10:      end while
11:   else
12:      c1 ← rand() % size
13:      c2 ← rand() % size
14:      while c1 ≡ c2 do
15:         c2 ← rand() % size
16:      end while
17:   end if
18:   if utility(new) < orig_utility then
19:      solution ← new
20:   end if
21: end for
22: if utility(solution) < orig_utility then
23:    return solution
24: end if
```

participant of the final deployment solution. For instance, if a user-provided dataflow contained five participants, then five chromosomes would exist in the solution list and the genetic algorithms will attempt to optimize the deployment by mutating chromosomes until a time limit is reached. The best generated solution that contained the lowest summed latency according to the edges in the user-provided dataflow would be returned by these genetic algorithms as the solution list (this list is actually a vector in MADARA for performance reasons).

Both algorithms select chromosomes (*i.e.*, nodes/drones) of the proposed solution (the approximated deployment) to mutate and then perform mutations for a specified time interval or number of allowed mutations before returning the best solution (either the original or the improved solution). The Guided GA in Algorithm 3, however, targets the higher degreed nodes 80% of the time and selects from the best available participants in the underlying cloud, which allows it to make more intelligent mutations by targeting highly degreed chromosomes more often. While the Guided GA does converge much more quickly than the Blind GA, the randomness inherent in the Blind GA can be better for hybrid approaches (solutions that combine Blind GA with other heuristics)

IV. Experimental Validation of the Heuristics

Due to space restrictions of the short paper format, we must present a limited glimpse into the significant optimizations made possible by these heuristic approaches. The specific type of experiment we will focus on here is called a system slowdown experiment. System slowdown is defined by the equation "slowdown = 2 * system_latency / (1,000,000 * size)" in these experiments. With the optimal configuration, slowdown == 1. Anything greater than one is a factor of slowdown. For example, 2.0 is a 100% slowdown in the overall system, which drain a battery more significantly than an optimal deployment. Our objective in the heuristics in this experiment is to reduce the system slowdown to 1, if at all possible.

In the experimental results shown in Figure 4, three specialized drones are collecting and instrumenting three disjoint segments of a 10,000 drone deployment within a noisy environment. In this environment, we have many local minima in the system slowdown but only one optimal 10,000 drone deployment.

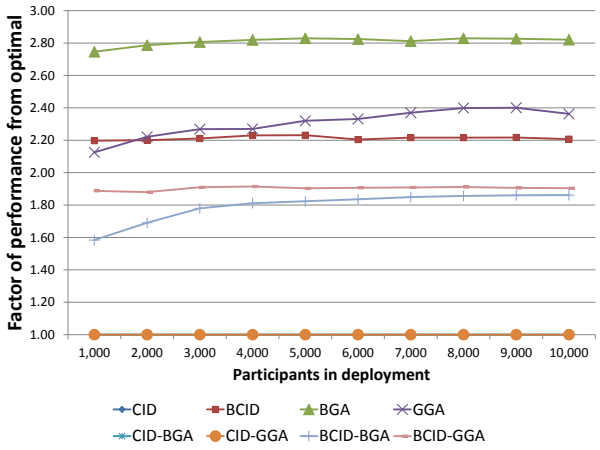

Fig. 4. System Slowdown with 3 Specialized Drones in a Noisy Environment

There are three important results that come from these graph. First, a random solution, which corresponds to the BGA line, is the worst performer and results in nearly 3x worse performance than optimal. Second, all CID-based algorithms (CID, CID-BGA, and CID-GGA) find the optimal deployment for 10,000 drones and do so within milliseconds. Third, the anytime algorithms BCID-GGA and BCID-BGA can reduce the system slowdown to less than 2x worse than optimal, which is significantly better than random.

V. Related Work

This section compares our work on MADARA with related work on deployment problems based on constraint satisfaction problem solving, genetic algorithms, and heuristics.

Constraint satisfaction problem solving. Haldik et. al. [9] presents a constraint programming technique to solve static allocation problems in real-time tasks. Cucu-Grosjean et. al. [4] propose two approaches to addressing real-time periodic scheduling on heterogeneous platforms, which is a constraint satisfaction problem (CSP). The first method requires an encoding of the problem into a basic format that is then passed into state-of-the-art CSP solvers. The other approach encodes problems in an optimized way to obtain solutions

faster. Despite being faster than traditional CSP solvers, both techniques take dozens to hundreds of seconds to solve even small number of constraints, which does not meet the requirements of our motivating DRE application in Section II that exhibits thousands of constraints defined on the deployment between the collection drone and its group.

Genetic algorithms. Whereas CSP solving typically involves backtracking through potential matches, genetic algorithms are a type of local search that tries to approximate an optimal match through mutations, fitness functions, and crossbreeding best candidates according to the fitness criteria. Heward et. al. [8] recently used genetic algorithms to optimize configurations of monitors in a web services application. This method is unsuitable for our motivating DRE application, however, since it requires roughly an hour to compute an approximated good configuration.

Wieczorek et. al. [16] use a genetic algorithm to schedule scientific dataflows in Grid environments, but their mutation-based scheme similarly required at least hundreds of seconds (some of their tests showed requirements of tens of thousands of seconds—several hours). Other implementers have used combinations of genetic algorithms and neural networks [11] and even knowledge and reasoning [10] to converge to optimal solutions. These approaches concentrate on offline or human-interactive solutions, however, and thus are not suitable for DRE application problem solving because they require many minutes or hours to approximate a solution.

Heuristics. Heuristics approximate good solutions and often serve as guides for local search techniques, such as genetic algorithms, simulated annealing, or backtracking and depth-first searches. Some researchers use these heuristics to directly approximate scheduling [8] in grids and dataflow solutions [3] for real-time solutions. The latter is of interest to us since the heuristic approximates a constraint problem involving a set of dataflows within milliseconds. The solution [3], however, was demonstrated on only five hosts and not thousands, so it is not readily apparent how to migrate our motivating DRE application to the heuristic defined in either of these papers.

VI. Concluding Remarks

Enterprise DRE systems are increasingly essential in mission-critical domains, such as aerospace, defense, telecommunications, health care, and financial service. This paper presented two heuristics provided in MADARA to approximate user-provided dataflows in next-generation DRE clouds. We also presented MADARA's genetic algorithms and hybrids of the heuristics and genetic algorithms to improve the solutions generated by the heuristics. We analyzed the results of experiments to validate the MADARA heuristics and genetic algorithms, as well as highlighted issues with unguided genetic algorithms in a representative DRE application context.

C++ code for the MADARA heuristics and algorithms is available in open-source form from madara.googlecode.com.

Acknowledgments

This work was supported in part by NSF SHF/CNS Award 0915976 and NSF CAREER Award CNS 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The views and conclusions contained in this document are solely those of the individual creators and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the Software Engineering Institute, or its Sponsors.

References

[1] K. Arisha, M. Youssef, and M. Younis. Energy-aware TDMA-based MAC for sensor networks. *IEEE IMPACCT*, pages 21–40, 2002.

[2] J. Balasubramanian, S. Tambe, B. Dasarathy, S. Gadgil, F. Porter, A. Gokhale, and D. C. Schmidt. Netqope: A model-driven network qos provisioning engine for distributed real-time and embedded systems. In *RTAS' 08: Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 113–122, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[3] T. Cucinotta and G. Anastasi. A heuristic for optimum allocation of real-time service workflows. In *Service Oriented Computing and Applications, 2011. SOCA '11. International Conference on*, pages 169–172, 2011.

[4] L. Cucu-Grosjean and O. Buffet. Global multiprocessor real-time scheduling as a constraint satisfaction problem. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pages 42–49, sept. 2009.

[5] J. Edmondson and A. Gokhale. Design of a scalable reasoning engine for distributed, real-time and embedded systems. In *Proceedings of the 5th International Conference on Knowledge, Science, Engineering and Management (KSEM)*, 2011.

[6] J. Edmondson, A. Gokhale, and S. Neema. Automating testing of service-oriented mobile applications with distributed knowledge and reasoning. In *Proceedings of the Service-Oriented Computing and Applications (SOCA)*, 2011.

[7] J. Elson and D. Estrin. Sensor networks: a bridge to the physical world. pages 3–20, 2004.

[8] G. Heward, J. Han, J.-G. Schneider, and S. Versteeg. Run-time management and optimization of web service monitoring systems. In *Service Oriented Computing and Applications, 2011. SOCA '11. International Conference on*, pages 294–299, 2011.

[9] 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*, 81(1):132–149, 2008.

[10] 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*, volume 5, pages 4350 – 4355 Vol.5, april-1 may 2004.

[11] A. Javadi, R. Farmani, and T. Tan. A hybrid intelligent genetic algorithm. *Advanced Engineering Informatics*, 19(4):255 – 262, 2005.

[12] J. S. Kinnebrew, W. R. Otte, N. Shankaran, G. Biswas, and D. C. Schmidt. Intelligent resource management and dynamic adaptation in a distributed real-time and embedded sensor web system. In *Proceedings of the 2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, ISORC '09, pages 135–142, Washington, DC, USA, 2009. IEEE Computer Society.

[13] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Real-time Systems. Springer, 2011.

[14] A. Sinha and A. Chandrakasan. Dynamic Power Management in Sensor Networks. *Smart dust: sensor network applications, architecture, and design*, page 1, 2006.

[15] V. Subramonian, G. Deng, C. Gill, J. Balasubramanian, L. Shen, W. Otte, D. Schmidt, A. Gokhale, and N. Wang. The design and performance of component middleware for QoS-enabled deployment and configuration of DRE systems. *The Journal of Systems & Software*, 80(5):668–677, 2007.

[16] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34:56–62, September 2005.