AIR - A Framework for Exploiting Redundancies in Service-Based Applications

Ulrich Kremer and Liu Liu
Department of Computer Science
Rutgers University
(uli,liu.liu)@cs.rutgers.edu

Technical Report DCS-TR-720, October 2015

Abstract

Redundancy is a fundamental property of most complex systems that makes them more resilient and productive. Instead of relying on a single solution to a problem, multiple solutions are feasible, each solution with its specific resource requirements and cost/quality tradeoffs. This paper targets service-based applications that consist of webs of collaborating services, where productivity, resilience, and energy consumption are crucial. Examples of such applications include mission software for autonomous robots (drones) and service-based smart phone applications.

This paper discusses a new framework that allows the specification and optimization of redundancy along three orthogonal redundancy dimensions, namely **A***pproximation,* **I***mplementation, and* **R***eplication (AIR). The RSDG (Redundant Services Dependence Graph) is a directed, dual-weighted graph with nodes representing redundant services and their quality levels, and edges representing service dependencies. For a given application, the* mission value *(productivity) of a service node reflects its importance for producing the desired overall application outcome, while the* energy value *represents its energy consumption, which is not application dependent. Given some application constraints, an optimal solution to an RSDG service selection problem is a set of nodes and edges that maximizes the total mission value while minimizing energy consumption. The optimal selection problem is shown to be NP-complete.*

AIR's compiler maps instances of RSDG service selection problems into two dependent 0-1 integer programming problem instances. Experimental results are reported for a service-based application that specifies sensing, actuation, and communication activities of an autonomous underwater vehicle (AUV). The results were based on an AUV deployment in the Atlantic Ocean. They show that the AIR framework is expressive and efficient. The overall application productivity and resilience was greatly improved by dynamically recomputing the optimal solution to reflect changes in energy reserves, mission goals, or to deal with service failures. In an application's design stage, AIR can be used to assess and adjust service redundancies for productivity and resilience under anticipated usage patterns and failure conditions. To the best of our knowledge, AIR is the first system that represents and optimizes all three dimensions of redundancy within a single framework.

1. Introduction

Redundancy is a fundamental property of most complex systems that occur in nature as well as in computer science. Redundancy makes systems more resilient, robust, and productive. If one component fails or is temporarily not available, another component or a combination of other components may be used to replace the failed one, but not necessarily with the same quality outcomes. Nearly every day, we are exploiting redundancies of some form, for instance, when finding another route to work if traffic is backed up, when replacing a flat tire with a spare, or when being forced to eat with our left hand due to an injury to our right hand, assuming we are right handed. Actually, without genetic redundancy, all of us would probably not even be alive. In a nut shell, redundancy provides crucial options to do the same thing in different ways, each with its specific cost/quality/resource tradeoffs.

This paper targets applications that are implemented as webs of interdependent services with significant redundancies. Each service may produce outcomes/answers of different quality. Instead of only a single possible answer, these applications may provide outcomes of different qualities depending on the quality of the data itself and the quality of the data processing services operating on the data. Applications may contain services that a particular execution may not require because the quality of the data does not match the input/ouput requirements of the particular service, or because another service can be used to provide the same output data in some preferable way. Each application execution has a set of constraints with respect to (a) the quality of the input data, (b) the required quality of the produced output data, and (c) the resources that may be used. For each application execution the challenge is to select the best set of services among all application services and their associated quality levels such that all constraints are respected while resources are minimized. The new AIR framework characterizes redundancy among dependent services along three orthogonal dimensions, **A**pproximation, **I**mplementation, and **R**eplication.

Approximation - Multiple implementations of the same service produce outcomes/answers of different qualities. Approximation as a technique to reduce energy consumption and/or improve response times has been discussed in many recent related works. By producing less accurate or fewer answers,

Table 1: Redundancy vs. Productivity/Resilience

Redundancy	Productivity	Resilience
Approximation	Only produce an answer of a required quality, not more, even if resources would be available (don't do more than needed).	If a lower quality answer is acceptable, choose it in the presence of resource limitations or failures (better less than nothing).
Implementation	Different implementations use different resources which may be shared to improve overall productivity.	In the presence of resource limitations or service failures, system is able to switch to implementation that still works.
Replication	Reproducing the same results locally instead of remotely, if cost is less than cost of communication.	If one clone fails, just switch to another that still works.

significant energy can be saved. Whether an answer is still acceptable in terms of its quality/performance is left to the application programmer's accuracy/performance needs. Examples for approximate services include computing answers of lower precision (fewer bits to represent a floating point number, or a coarser grid used for a discrete fluid dynamics application), or producing fewer answers (fewer reported results for a web search, fewer GPS readings in an outdoor localization service, or fewer frames per second used for streaming a video). It is important to note that in its basic form, an approximation uses the same algorithm and same resources as the "precise" version, but "relaxes" some aspects of these algorithms and resources to achieve the approximation.

Implementation - Multiple implementations of the same service produce the same outcomes/answers of the same quality, but using different algorithms and/or resources. This approach has been used, for instance, in software engineering to build systems that can deal with program bugs. Relying on different types of hardware resources to run an application also falls into this category.

Replication - Multiple identical copies of the same software implementations or hardware components. The basic idea is to do the same thing utilizing different, but functionally identical resources (clones). This is perhaps the most basic form of redundancy. Examples include the spare tire in a car (full size spare - otherwise it is an approximation), the five space shuttle computers, and the design behind the Tandem computer system. Replication can have many benefits, including increased fault tolerance and reliability. It also can save energy and other resources in cases where generating a particular outcome/answer is cheaper than communicating that outcome/answers to the services that need it.

The three different aspects of redundancy can lead to systems with highly complex dependencies and interactions between their provided services. For example, approximations can use different implementations at different service quality levels. Also, a services may provide a necessary input to several other services, allowing the cost of producing its output data to be amortized across all the services that require the data as input. This can lead to a non-monotonic cost/quality behavior, where it may be "cheaper" for some services to produce an answer of higher than of lower quality. Using data from a replicated service may have an additional communication cost that depends on how "close" the clone is relative to the service that needs its data. Finally, producing an overall outcome of a desired quality may change the quality requirements of all the services the outcome depends on, resulting in a system that is able to balance and adjust the quality and selection of services across the entire service web in response to changing quality requirements, service availabilities (failures), or energy budgets.

The AIR framework models these complex interactions within a single framework, and allows the static (prior to execution) or dynamic (during execution) selection of the best service configurations for a given quality requirement under specific resource constraints, particularly energy constraints. Previous work on redundancies mainly concentrated on reliability and robustness. However, the different dimensions of redundancies may also be used to improve the productivity on an application. Some potential benefits of the different redundancy dimensions are listed in Table 1. The main contributions of this paper are:

- the representation of three orthogonal dimensions of redundancy within a single optimization framework in a dual-weighted graph (RSDG). Computing the optimal solution is shown to be NP-complete;
- the 0-1 integer programming formulation and solution of a dual-objective optimization problem based on mission value (productivity) and energy (resources) metrics. Mission values allow users to specify intra-service and inter-service priorities, i.e., which services and associated quality level are important for the particular application execution;
- a framework that can be used statically or dynamically to evaluate/reconfigure an application for productivity and resilience along all three redundancy dimensions; and
- the evaluation of the new AIR framework based on a realistic application which controls the sensing, communication, and actuation activities of a marine robot. The experimental results are based on a deployment of the marine robot in the Atlantic ocean.

The experimental results show that the AIR framework

is effective and efficient. For the marine robot application, the AIR produced solutions with mission value improvements of over 1100%, with average improvements of 18.2% and 25.6% over two current heuristics. Productivity and resilience were also enhanced by allowing users to experiment with different mission value settings and redundancy characteristics for their applications, enabling them to assess the cost/energy/productivity/resilence tradeoffs through a tight feed-back loop. An efficient 0-1 formulation combined with memoization strategies allow dynamic reconfigurations of the application services in the presence of environmental uncertainties and service failure conditions, improving the productivity and resilience of the application.

The remainder of the paper is organized as follows: Section 2 discusses the foundations of our new AIR framework with its RSDG represntation. The illustrating application targets the Slocum glider and is presented in Section 3. Section 4 on experimental results is followed by a discussion of related work. A summary of the results and future work conclude the paper.

2. Foundations of the AIR Framework

The AIR framework uses the **R**edundant **S**ervices **D**ependence **G**raph (RSDG) as its main representation for service-based applications. The RSDG is a directed graph with node and edge weights. Nodes represent services or groups of services. A node that represents groups of services is referred to as a layered node. Non-layered nodes are called basic nodes. Edges represent producer/consumer (data flow) dependencies between two services and/or groups of services, with information flowing from the source of the dependence to the sink of the dependence.

The RSDG is a dual-weighted graph where nodes (services) may have two distinct weights. The *mission value* weight of a node respresents the importance of the associated service for a particular application execution. Mission value weights may be assigned to layered and basic nodes. The second node weight represents the *energy* consumption of the node (service) if active as part of an application's execution. Only basic nodes have energy weights. RSDG edges may have energy weights reflecting the energy costs of transmissing the data, or performing data manipulations such as interpolation ("quality upgrade") or down-sampling ("quality degradation") to satisfy the data quality and format requirements of the sink node (data consumer) of the dependence.

Figure 1 shows an example RSDG representing five different services and their dependencies. All three dimensions of redundancy are represented in this illustrating example. Approximations are represented by a vertical list of nodes with a descending order of quality. In the example, services S_1, S_2, and S_3 have two approximation levels each. The highest quality service of S_1 has two possible implementations. This is represented by a horizontal list of nodes. Representing replication does not need any particular node structure. A

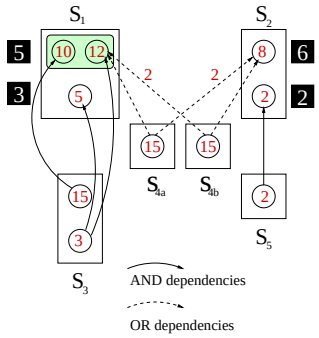
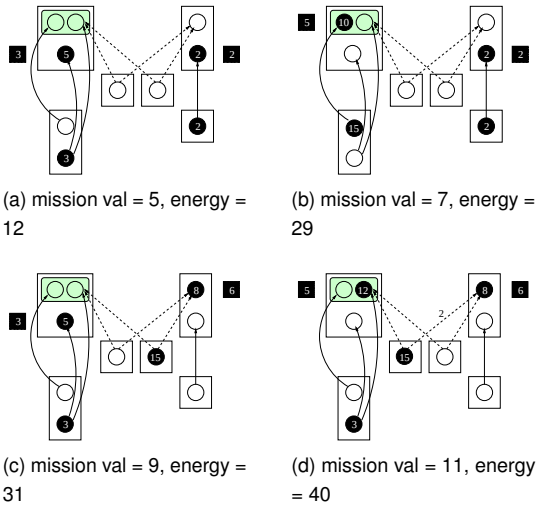


Figure 1: Example RSDG with 5 services. Nodes and edges have edge weights reflecting their energy requirements. Critical services S_1 and S_2 have mission value weights for their approximations. All three dimensions of redundancy are represented.

replicated service is represented as a set cloned nodes with their dependencies. In the example, S_4 has two replicated copies represented by nodes S_{4a} and S_{4b}.

Edges represent dependencies between nodes and their associated services. If a node is selected to be part of an application's execution, the nodes it depends on have to be selected as well. The RSDG models two types of dependencies, *AND* and *OR* dependencies. *AND* dependencies allow a service to be dependent on a set of different services which are all needed to accomplish the service's task. Sources in a group of *AND* dependencies with the same sink node have to be distinct services. In other words, two *AND* dependencies cannot originate from different nodes in the approximation list for the same service, or different nodes in the implementation list of the same service. The reason for this restriction is that the final node selection for an RSDG requires a service to be respresented by at most one of its basic nodes. It is important to note that replicated services are modelled as distinct services in the RSDG, and therefore multiple replicated services could be active during an application's execution. *OR* dependencies allow a service to be dependent on a set of alternative services and/or alternative approximations or implementations of a service. *OR* dependencies are grouped, where a single source node has to be selected from each *OR* group. The current AIR framework does not allow *AND* and *OR* to be nested. However, more complex dependency relations can be expressed using "dummy nodes" in the RSDG which do not have any associated services in the application. In order to simplify the discussion in this paper, we will limit the dependencies to single-level *AND* or *OR* dependencies, i.e., the different types of dependencies cannot be nested. Figure 1 has *AND* and *OR* dependencies. For example, if the second implementation of the highest quality approximation of service S_1 is selected for execution (node weight 12), then the lowest quality of approximation of service S_3 has to be selected (node weight 3), and either service S_{4a} or S_{4b} have to be selected (both node weights 15, but different edge weights).



(a) mission val = 5, energy = 12

(b) mission val = 7, energy = 29

(c) mission val = 9, energy = 31

(d) mission val = 11, energy = 40

Figure 2: Optimal solutions with their maximal mission values and minmal supporting energy budgets.

The RSDG is a representation of an application as a web of interdependent services and their redundancies. An **RSDG node selection problem** requires as input an RSDG, a set of *critical* services, i.e., services that the application user wants the application execution to provide, and an energy budget. A **solution of the RSDG node selection problem** selects a basic node for each critical service, and a set of supporting nodes that need to be selected due to dependencies. The selected nodes have to satisfy the constraints that (a) only a single basic node for each service can be selected, and (b) the sum of the edge and node weights of selected nodes has to be less or equal to the specified energy budget. For our sample RSDG in Figure 1, a user may specify S_1 and S_2 as the critical services. Figure 2 shows possible solutions for this specific problem.

One important new feature of the AIR framework is that the user can specify the relative importance of approximation levels of critical services (intra-service priorities) and the relative importance of different critical services (inter-service priorities) using the *mission value* metric. By definition, non-critical services are supporting services only and therefore are not assigned any priorities. Figure 1 shows an instance of the RSDG selection problem with S_1 and S_2 as the critical services, and with mission value weights for the two approximations for each of the critical services. Mission values model intra-service productivity across the different approximation levels (e.g.: 5:3 for S_1 and 3:1 for E_2) and inter-service productivity across different services and their levels (e.g.: the lower quality approximations for service S_1 has 50% of the importance of the highest approximation level for service S_e modelled by the ration of 3:6). A **maximal solution of the RSDG node selection problem** is a solution to the RSDG node selection problem that maximizes the sum of selected nodes' and edges' mission values, i.e., maximizes for application productivity while respecting the energy constraint. An **optimial solution of the RSDG node selection problem** is a maximal solution that also minimizes the energy consumption under the given energy constraint. In other words, the optimial solution tries to first find the most productive solution under the energy budget, and then, if there are several such solutions, pick the one with the lowest energy consumption. It is important to note that a maximal solution does not exclude situations where services are selected which produce results that are not needed to support any critical services, as long as the energy budget is respected. In addition, there may be different sets of supporting services with energy requirement that are different, but that all fit within the specified energy budget. Figure 2 shows the optimal solutions for different sample energy budgets.

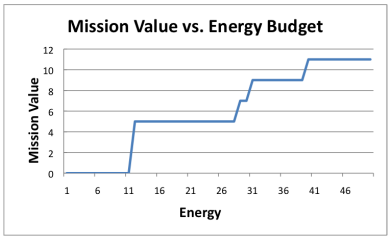


Figure 3: Productivity Profile for sample problem of Figure 1.

The RSDG represents a rich optimization space where semantic properties such as mission productivity and system properties such as service dependencies, energy consumption, and redundancies can be expressed and optimized in a unified framework. Optimal solutions may change in responds to changes in the set of critical services and their mission values, availability of services, and energy budget constraints. For our sample RSDG, optimal solutions for a changing energy budget exploited all three dimensions of service redundancies, supporting the need for modelling all three redundancy aspects with a unified representation. Figure 3 shows the relation between a provided energy budget and the maximal mission value that can be achieved for that budget. Reducing the energy budget of an optimal solution will result in a decrease of the mission value. Some optimal solution seems initially non-intuitive, for instance the quality improvement of S_1 in response to a decrease in the energy budget from 31 to 30, with a resulting decrease of the mission value from 7 to 5. However, this is possible since energy is optimized across all services active during application execution.

The information presented in Figure 3 can also be used as a productivity profile of an application. It is not only important for a user to understand the productivity that can be achieved under a given energy budget, but also where the transitions between overall application qualities occur. For our sample application, an increase of the energy budget from 12 to 29 (240%) is needed to raise the productivity from 5 to 7 (40%). However, to raise the productivity from 7 to 9 (29%), the

energy budget has to be increased from 29 to 31, which is 7% of the budget. Intra and inter-service priorities expressed as mission values are not absolute values, but are meant to express relative priorities. We expect situation where a user of a service-based application will need to calibrate these relative priorities by "playing" with different mission value assignments and observing their impact on the productivity profile. If the productivity profile does not have the desired shape, available redundancies in the system may need to be changed in any of the three redundancy dimensions. The productivity profile may also significantly change due to failure conditions. A user can assess the resilence of an application by observing the impact of different service failure scenarios on the productivity. For our sample application, a failure of S_{4b} will not impact the optimal solution under an energy budget of 40 or more. However, it will increase the energy requirements of the optimal solution with 9 as its mission value. S_2 has to switch to S_{4a} to provide the needed service, which inceases the minimal supporting energy budget from 31 to 33. The energy requirements of the remaining optimal solutions do not change due to the failure. Using the AIR framework in a pre-mission testbed environment allows users to evaluate the productivity and resilience characteristics of their service-based application. If a user is not satisfied with the outcome of the assessment, inter or intra-service priorities may need to be adjusted, or redundancies of services need to be changed. Another important function of the AIR framework is the dynamic adjustment of service selections due to unexpected resource conditions (e.g.: unexpected loss of battery energy) or service failures. Instead of aborting an application, the application maintains productivity under the changed service and/or energy availabilities, if possible.

In the following, we will first show that the optimal service selection problem is NP-complete before presenting our problem solution.

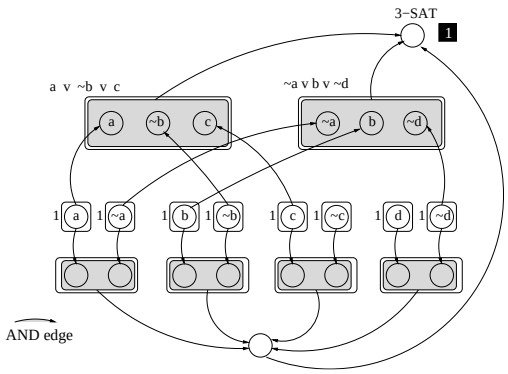


Figure 4: Translation of a 3-SAT instance consisting of two terms, $(a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg d)$ into an $RSDG_{translate}$.

Proof of NP-completeness of the optimal RSDG node selection problem An instance of the *3-SAT* problem can be translated into an instance of the RSDG problem with only *AND* edges in polynomial time. The instance of the *3-SAT* problem is satisfiable if and only if the energy budget of the optimal solution of $sol(\{3SAT\}, RSDG_{translate}) = |\text{VARS}|$, where VARS is the number of literal variables in the conjunctive normal form and $3SAT$ is the only critical node in $RSDG_{translate}$.

A variable or its negation is referred to as a literal. Each 3-SAT term (disjunction of three literals) is represented by a service with an implementation list of exactly three service alternatives, each service associated with a single literal in the term. Each variable that occurs in the 3-SAT formula is represented by two services of weight 1, one representing the variable, and one its negation. All other node and edge energy weights are 0. The mission value of the only critical node is 1. The translation is illustrated in Figure 4.

"⇒": *3-SAT* has a solution $\Rightarrow sol(\{3SAT\}, RSDG_{translate}) = |\text{VARS}|$: If there is a solution to the *3-SAT* problem, then there has to be an assignment of variable to truth values that makes the formula true. This means that there has to be a corresponding selection of the service for each variable or its negation which will allow each service that represents a term to select a literal that makes the term true. The translation also introduces service nodes that require at least one of the services representing a variable and its negation to be selected. Therefore, the minimal solution to the RSDG instance exists and has an energy cost of |VARS| since each variable or its negation has to be selected.

"⇐": $sol(\{3SAT\}, RSDG_{translate}) = |\text{VARS}| \Rightarrow$ *3-SAT* has a solution : Every solution of an $RSDG_{translate}$ instance has to select a service for each variable, either negated or not negated. Since the minimal solution is n, a variable and its negation cannot be selected at the same time (mutual exclusion) since this would result in a cost greater than n. Since a solution exists, each term has to be able to select a service that represents a literal that makes the service true. Since the *3SAT* is critical it has to be selected. This implies that all mutual exclusive services (bottom row in Figure 4) and all services representing 3-SAT terms have to be selected. This means that the selection of the n services associated with the variables or their negation has to satisfy the represented instance of the *3-SAT* problem.

This concludes the proof together with the fact that a given solution for an instance of the selection problem can be verified in polynomial time. □

0-1 integer programming formulation Instances of the optimal RSDG node selection problem can be computed efficiently by mapping them to equivalent 0-1 integer programming problems. The translated formulations can then be solved by general purpose integer programming tools such as CPLEX [3], Gurobi [20], and lp_solve [21]. A 0-1 integer programming formulation consists of variable declarations, a set of constraints, and a single objective function that may be either minimized or maximized. Let $var(x)$ denote the unique 0-1 variable associated with either a node or edge x in the

(1) Set $var(c)$ = 1, i.e., select all critical nodes
(2) Maximize the **mission value** objective function

$$\sum_{x\ in\ RSDG} mission\_value\_weight(x) * var(x)$$

under the feasible solution constraints, and the energy constraint

$$\sum_{x\ in\ RSDG} energy\_weight(x) * var(x) \leq Energy\_Budget$$

(3) Set $var(s)$ = 1, where s has been selected as a basic node of a critical service in the above maximization solution. Note: This forces energy minimization to have the optimal mission value.
(4) Minimize the **energy** objective function

$$\sum_{x\ in\ RSDG} energy\_weight(x) * var(x)$$

under the feasible solution constraints

Figure 5: Algorithm for optimal RSDG node selection.

RSDG. A node/edge x is said to be selected if $var(x)$=1, and not selected if $var(x)$=0. The following constraints encode the structure of the RSDG and the properties of a feasible solution.

- *AND edges*: If a sink node of a dependence edge is selected, the source nodes has to be selected as well.

  $$var(sink\_node) <= var(source\_node)$$

  Note: If the sink is not selected, the source may or may not be selected.
- *OR edges*: If a sink node of a group of OR dependence edges is selected, exactly one source node of the group of edges has to be selected. This can be expressed with a set of matching IN-constraints and OUT-constraints as discussed in [4].
- If the layered node is an implementation list or list of approximations, only a single implementation or approximation may be selected.

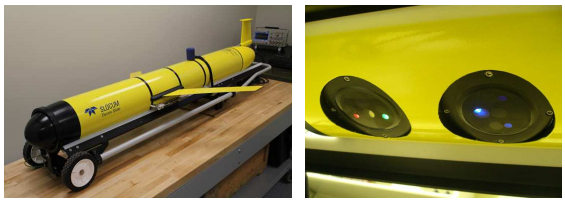
  $$var(node_{layered}) = \sum_{n \in list} var(n)$$

  Note: If the layered node is not selected, then all of its component nodes cannot be selected.

Let c denote a node representing a critical service. Figure 5 gives a sketch of the algorithm used to solve the optimal RSDG node selection problem as two consecutive integer programming problems.

3. AUV With CTD and ECO-Puck Sensors

Autonomous underwater vehicles (AUVs) such as the Slocum Glider [13] have revolutionized the way physical oceanographers and other marine scientists can learn and understand the processes within our oceans and how they affect marine life. The mission management application for a marine robot



(a) Slocum Glider (b) ECO-puck sensors

Figure 6: (a) Our Slocum Glider equipped with a double payload bay is 180 cm long and weighs around 70 kg. (b) WET Labs' BB2FLSV4 and BB2FLSV5 backscatter and fluorescence sensors installed on our Slocum Glider.

will be used to illustrate the expressiveness and efficiency of the AIR framework. Our Slocum Electric Glider shown in Fig. 6a is a buoyancy driven AUV developed and produced by Teledyne Webb Research [19]. There are over 400 Slocum gliders in use today, and together with other gliders [14, 7, 9], they arguably represent the largest class of autonomous marine robots in operation today. The front of the Slocum Glider contains a buoyancy engine which moves a piston to change its displacement of water, allowing for vertical motion in the water column. The pitch of the glider can be adjusted by moving an internal battery pack, thereby changing its center of gravity. The AUV's wings allow it to glide forward through water to produce a saw-toothed flight profile inflecting near the surface and at deeper depths. Satellite and radio communications are used at the surface to transmit data and alter, if necessary, the AUV's mission [13]. Our Slocum Glider contains two 16 Mhz computing platforms [11] and a single board 200MHz ARM computer.

Battery operated AUVs have many operational characteristics in common with mobile platforms such as Smartphones. Both have a large variety of sensing, actuation, computation, and communication capabilities, all of which have to be supported by battery energy. In 2009, a Slocum glider flew from New Jersey to Spain in 221 days, making it the first robot crossing the Atlantic Ocean [8]. This mission also demonstrates the danger of glider operations. An attempt to cross the Atlantic a year before ended when the glider was lost at sea after a few months into the mission. It is quite possible that better redundancy management could have saved the mission.

The AIR framework uses the RSDG to model the services and their dependencies. Our mission management sample application has four major classes of services as shown in Figure 7. Location services, both at the surface and underwater, are important for navigation and for the collected spatiotemporal data. Any acquired data is typically tagged with a location coordinate and a time stamp. The sensing services can be very complex depending on the particular sensor payload of the AUV. In our case, the application specifies a main sensor payload of two CTD sensors and two ECO-pucks. The CTD sensors measure the conductivity (salinity), temperature of the water, and the diving depth (water pressure). The ECO-

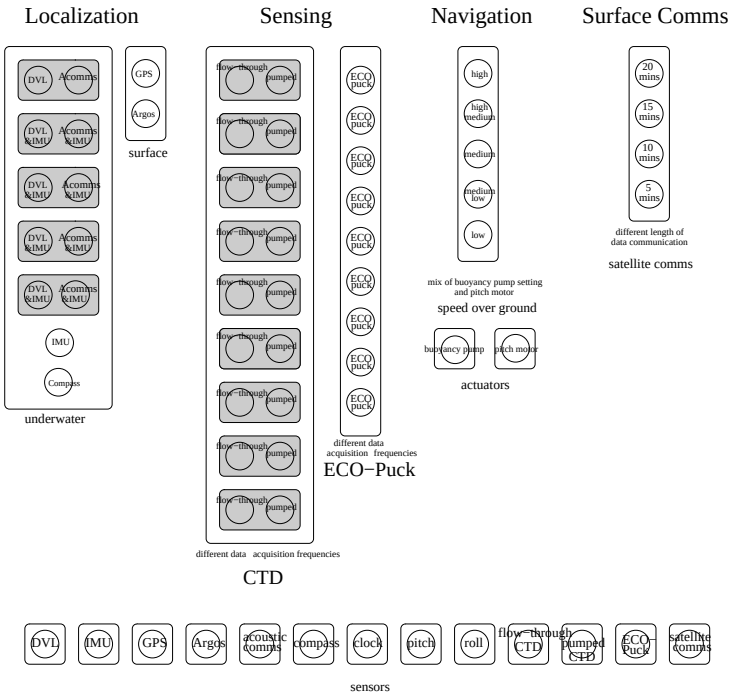


Figure 7: A (partial) RSDG for a system of services for an autonomous underwater vehicles (AUV) during a flight segment. The glider is equipped with a flow-through and pumped CTD (conductivity, temperature, and depth sensor), and two ECO-pucks (backscatter and fluorescence sensors). Node weghts and the over 400 dependence edges and their weights are not shown.

sensor / motor	explanation	power in Watts
ADCP (DVL)	acoustic doupler current profiler	1.18
ECO-pucks	flurometer	0.75
IMU	inertial measurement unit	0.02
compass	magnetic compass	0.01
GPS	global positioning	0.5
CTD	conductivity, temperature, depth	0.05
pumped CTD	conductivity, temperature, depth	0.25
buoyancy pump	change buoyancy	3 - 60 (depending on depth)
pitch motor	changes center of buoyancy	2
WHOI's micro modem	acoustic underwater communication	10
Iridium	satellite surface communication	4.5
Argos	emergency communication	0.5

Figure 8: AUV: Measured and estimated power dissipation of selected sensors and motors.

pucks (environmental characterization optics) can be used as flurometers or for volume scattering. Actuators provide the propulsion needed for navigation, which in the glider case are the buoyancy pump and the pitch motor. While the glider is at the surface, it can upload and download data through a satellite communication link. Typically, a small subset of the acquired sensor data together with some glider health data is uploaded at each surfacing. The power requirements of some of the deployed sensors, actuators, and communications are listed in Table 8.

Localization: For AUVs there are different localization technologies available for surface and underwater localization. While on the surface, satellite (radio) communication is available to access GPS (Global Positioning System) information, or connect to the Argos (Global Localization and Tracking) system. Typically, GPS provides a better resolution as compared to the Argos system. Since radio communication is not effective underwater, the GPS and Argos services are not available when the AUV is submerged. Typically, a GPS or Argos reading is taken just before a dive. This reading is then used as a reference point for underwater localization techniques. The most precise technique uses a DVL (Doppler Velocity Log) to track the ocean bottom or layered ocean currents. The relative motion above the ocean floor together with the depth readings gives the position of the AUV in the 3-D space. Since running the DVL is expensive, the next two approximations measure the absolute speed-over-ground using different time intervals, while "filling the gaps" with the IMU (Inertia Measurement Unit). The IMU is used for dead reckoning, but cannot account for cross ocean currents which may be different at different diving depths. Running the DVL at different time intervals allows to system to "recalibrate", resulting in a trade-off between energy consumption and localization precision. Similar localization precision can be achieved through triangulation via acoustic underwater communication. Therefore, both DVL/IMU and acoustic communication based technologies can provide the same service quality, but using different implementations (implementation redundancy). A cheaper methods may only utilize the IMU, avoiding the cost of the DVL or acoustic communication. Finally, our simplest approach uses just a compass, the dive angle, and diving time to determine the AUV's location. For our illustrating example as shown in Figure 7, we use 7 levels of approximations.

Sensing: The example system uses CTD sensors and a pair of WET Lab's ECO-pucks as its main payload sensors. Both flow-through and pumped CTD provide the same information

and therefore only one should be active at any given point in time. Although the flow-through CTD is cheaper to operate in terms of power requirements and energy consumption as compared to the pumped CTD, it has only a limited precision if the AUV speed falls below a particular threshold [2]. Therefore, the flow-through CTD requires a high or medium speed, while the pumped CTD can operate effectively at all speeds. The different approximation levels reflect the different discrete measurement frequencies. Measurement frequency is directly related to the power and energy efficiency. The fewer measurements are taken, the lower the spatiotemporal resolution of the sensor data. Again, we arbitrarily selected the number of approximation levels as 9. The two WET Lab's ECO-pucks are configured as a pair, i.e., operate as a single combined sensor. As the CTD, the ECO-pucks use 9 service quality levels.

Navigation: For gliders, vehicle speed is related to the buoyancy and dive angle of the vehicle. Different combinations of buoyancy and dive angle can be achieved by running the buoyancy engine and pitch motor for different lengths of time, consuming different amounts of energy. Again, to simplify our example, only 5 combinations of buoyancy engine and pitch motor activations are considered which induce different glider speeds. Since a slower glider speed will prolong the flight segment, more energy will be spent on data acquisition, and localization services since these activities are triggered by the glider's heartbeat. Service communication is also affected since typically a proportional sample of the acquired data is uploaded to a control center for inspection by the pilot and/or oceanographers. In order to reflect these additional costs, a fixed infrastructure cost is added to slower navigation speeds to approximate the additional energy consumption.

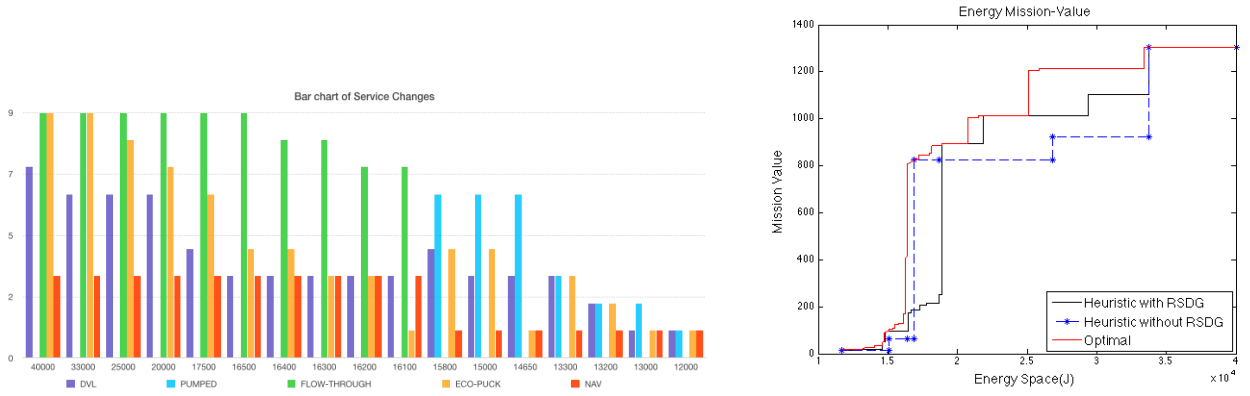
Surface communication: While the glider is at the surface, it typically contacts its control center via an Iridum satellite communication service. This allows glider pilots and oceanographers to download a subset of the scientific data acquired during the previous flight segment, check the glider health, and if necessary retask the glider by downloading new mission parameters. For our sample mission, we assume that pilots will download data proportionally to the amount that has been acquired during the last flight segment. Depending on the sensing quality (finer spatiotemporal resolutions), the satellite communication is assumed to take between 5 and 20 minutes.

All the above discussed dependencies between the four classes of services and their approximations and implementations have been encoded in the RSDG representation through appropriate service dependencies. The application RSDG also encodes additional dependences and restrictions. For sensing and localization services, only high resolution data needs high resolution localization information. This is expressed as OR-dependencies between sensing quality levels and localization services. For example, the five top approximations of the CTD / ECO-puck sensing services may only use any of the five top approximations of the localization services. In contrast, the lowest level of the sensing services may only use the two bottom quality localization services. Figure 7 does not show the service dependence edges, nor node or edge weights. The graph would be unreadable since it has over 400 edges.

4. Experimental Results

The experimental results were based on the deployment of our glider in the Atlantic Ocean on September 27, 2012. The glider flew at a dive angle of 26 degrees between the water depths of 5 and 60 meters. The test mission performed 3 yos (a yo is a pair of dive and climbs), and took around 50 minutes. For the reported experimental results, we assumed a flight segment that is three times as long with 9 yos and mission length of 2 1/2 hours, which represents a more realistic glider flight segment for week long missions. The RSDG as discussed in Section 3 represents the 2 1/2 flight segment with appropriate node weights to reflect the energy consumption of each basic service. This RSDG serves as the main basis for our experimental results.

For our first experiment, we assumed that the oceanographer specified the CTD and ECO-pucks sensing services as critical. The oceanographer also specified the intra-service priorities as linear, i.e., the mission values of the sensing service approximations are assumed to be directly proportional to the measurement frequency, and the CTD to have double the priority as compared to the ECO-pucks (inter-service priority). The resulting instance of an RSDG node selection problem was translated into a corresponding 0-1 integer programming problem by AIR's RSDG instance compiler. The compiler consists of approximately 400 lines of Scheme code. The compiler targets Gurobi, CPLEX, and lp_solve which have nearly identical format requirements for 0-1 problem instances. For the example RSDG, the compiler generated a 0-1 formulation with 557 variables and 775 constraints. The fastest tool, Gurobi, solved the problem in 0.05 seconds on average. These numbers were achieved on a multi-core laptop computer. We also experimented with different node and edge weights not representative for the flight segment, or chose different intra and inter priorities. In all cases, we obtained the same performance results. During a mission, a solution to the RSDG can be computed online, i.e., onboard, or off-line at the remote control center where the result is communicated back to the glider via a satellite link while the glider is at the surface. Our current gliders are equipped with a low-power ARM architecture which does not support CPLEX or Gurobi, but can run lp_solve. Although lp_solve is much slower than CPLEX or Gurobi (it took 60 seconds on average to solve the RSDG instances and other test instances), its power dissipation on the ARM was around 1 Watt, which translates into an energy cost of 60 Joules for a dynamic service reconfiguration. This is just a small percentage of even the lowest energy budget, and therefore is more than worthwhile doing due to the re-



(a) Optimal service selection under different energy constraints.

(b) Productivity profiles for three solution strategies.

Figure 9: (a) Services as part of the optimal service selection , and (b) Productivity profile of optimal solution together with provies of two heuristic solutions.

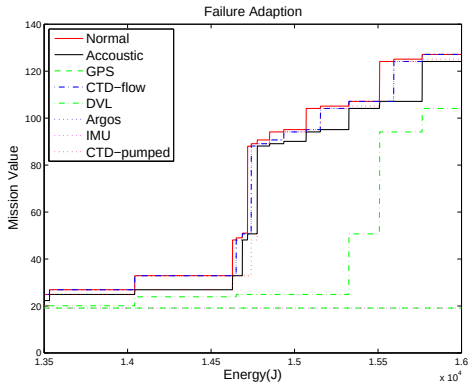
sulting energy savings and improvement in resilience after a service failure. In addition, using memoization, we were able to encode around 8000 service selection problem instances, including different service failure patterns, in a hash table, which was implemented on the ARM board. A single table look-up took less than 0.01 seconds, making memoization an effective scheme for anticipated, critical service selection instances.

Optimal Services and Their Approximations. Figure 9 shows the optimal solutions of the service selection problem under decreasing energy budgets. Figure 9a shows a subset of services (DVL, Pumped-CTD, Flow-through-CTD, ECO-puck, Navigation) and their approximation levels as part of the optimal solution. The higher the bar, the better the quality of the particular services. The graph illustrates that dramatic changes to the optimal service selection can occur in response to only a slight change in the energy budget. For example, when reducing the available energy from 16.1 KJoules to 15.8 KJoules, the optimal selection involves the slower glider speed, which in turn requires the pumped CTD to be activated since the slow speed cannot support the flow-through CTD. It is also important to note these different optimal solutions exploited implementation redundancies, namely the availability of the pumped and flow-through CTD. The optimal service selection is rather non-intuitive and hard to manage for an application user such as an oceanographer, motivating the need for the AIR framework. Figure 9b shows the corresponding mission values of the optimal solutions for the different energy budgets, i.e., the productivity profile (see also Figure 3). Mission values are only determined by critical services.

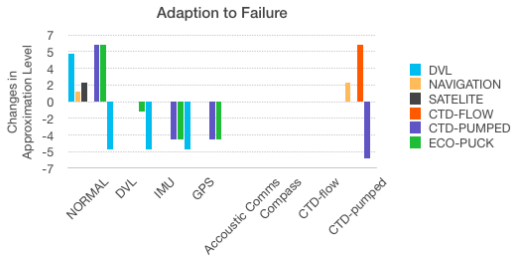
Comparison of Optimal and Heuristic Solutions. In addition to the productivity profile of the optimal solution, Figure 9b also shows the mission values computed by two heuristic. We used two heuristics to compare the optimal solution against. The heuristic without using the RSDG and its encoded dependencies uses three quality levels for all services. If the current service selection cannot be supported by a reduced energy budget, the quality of the most expensive service in terms of energy consumption is reduced. The second heuristic uses the RSDG, i.e., all its approximation levels and its dependences. Its strategy is similar to the first heuristic, but it will not select the most expensive service for a quality reduction if this would lead to an infeasible solution. As can be seen in Figure 9b, the optimal solution always produces a better result than the two heuristics. The heuristic using the RSDG does better most of the time, but there are situations where the simplest solution outperforms it. It is important to note the two stage approach of our optimal strategy was able to reduce the energy to support the solution with the highest mission value in some cases by up to 5%.

Based on our many year long collaboration with research groups that use gliders for their scientific missions, the first heuristic most closely matches the strategy that is used by oceanographers and pilots to deal with reduced energy budgets. In fact, just shutting down the most expensive scientific sensor is probably the most common approach, also because decisions are typically made rather quickly and in real-time, and therefore are conservative, while the glider is drifting on the surface. The conservative approach of glider pilots and oceanographers extends to avoiding remaining energy budgets that become too low since a sufficient energy reserve is needed to safely recover the glider if something goes wrong. In the range of 15,000 to 35,000 Joules, the optimal solution improves the average mission value by 18.2% over the non-RSDG heuristic, and 25.6% over the heuristic that uses the RSDG. The improvements range from 0% to 284% for the non-RSDG heuristic, and 0% to 1167% for the RSDG heuristic.

Failure Conditions. For a given energy budget, we forced a particular service to become unavailable. Expressing service failures is rather straight forward. A service failure is modeled



(a) Productivity Profiles Under Single Service Failures



(b) Service Selection after Single Service Failure, 16100 Joules

Figure 10: Adaptation of different system services to a single component failure; Figure 10a: Productivity Profiles after failures; Figure 10b: Changes in services and their levels (x-axis) relative to the optimal solution without the component failure.

by setting the corresponding 0-1 integer variable to 0. This forces the solver to pick an alternative solution that does not involve the failed service, if such a solution exists. Figure 10a shows the productivity profiles of our application under single service failures. This gives insight into the criticality of a service for the overall application productivity under different energy budgets. For example, losing the DVL service during a period of high energy reserves can be tolerated, but not at low energy levels since other alternatives are no longer available due to their energy demands. In contrast, losing acoustic communication does not significantly impact the overall productivity. However, a failed GPS or IMU can substantially limit the productivity of the application. Figure 10a shows the alternative service selections in the case of a single service failure for a specific energy budget, namely 11,000 Joules. Failed services are listed on the x-axis. The graphs show the drop or increase in service levels (y-axis) relative to the solution where the service did not fail. This graphs indicates the connection of the different services in terms of redundancy. For the failure of some components, such as the ECO-pucks, compass, clock, buoyancy pump, pitch motor, and satellite communication, no feasible solution was found.

Static Productivity And Resilience Analyses. Finding the solution to the optimal RSDG node selection problem is hard and requires a deep understanding of the application that even expert users may not have, let alone application users such as oceanographers or an average smart phone user for phone-based applications. Productivity profiles are not only application dependent, but also depend on the the energy budget, and intra and inter-service service priorities. The AIR framework has an important role in allowing application users to better understand the specific energy/productivity/resilience tradeoffs in their application. For example, productivity profiles of failure conditions can indicate whether a particular service has too much, or too little redundancies. As a result, the application can be adapted to better fit a user's need. A productivity profile can tell the application user what energy budget range is most sensitive to optimal service selection, Typically, simple heuristics work well in a high resource environment (all services can be selected), or very low resource environment (very few services can be selected). The challenge lies in the energy ranges with limited options, where picking one service at a particular quality level over another can make a significant difference for resilience and productivity.

5. Related Work

There has been substantial work in the area of replication and redundant implementations mainly for the purposes of building more robust and reliable systems. Here, we will concentrate on the most recent work on approximations. The novel feature of our AIR framework is the fact that it addresses system-wide redundancies across all of its dimensions of replication, implementation, and approximation, enabling the selection of tradeoffs that has not been possible before.

Language-based implementation: There have been a number of research efforts that focus on computation approximation, including new language based implementation as a tool to express the approximation of a program, like [12], which enables users to explicitly define an operation as Approximated or Precise, and [17], a LISP like language using a tree representation with several classes of nodes to describe the structure of a program and express the desired precision, enabling users to link [17] to normal C++ programs and choose among several paths to achieve the precision with a low cost. However, implementations that require explicit precision description like both [12] and [17] will always have the limitation that programmers should be fully aware of how different precision would impact the energy consumption. For example, in [12], programmers need to point out the exact operations that could be approximated, like a scheduler. It doesn't have the power of always being able to choose the optimal solution under various execution environments. And in [17], the language has the power of analyzing the program structure and find out the optimal solution, but only statically. Given a task that requires long execution time, like running an underwater vehicle for months, a language with explicit numbers of precision would

be no better than directly writing a lower cost implementation. But with the power of RSDG, when the environment changes, like shutting off a service node, the system could dynamically recalculate based on the new dependence graph.

Approximation computing algorithms and models: Besides the language based implementation, there has also been research on systems with new algorithms to locate the best solution point in the performance-consumption curve, the trade-off space. The main goal of these systems is to maximize the performance under energy restriction. These algorithms include [15], trading computation accuracy for program performance, and [10], which autotunes the program execution by adjusting crucial configurations. [22] provides a model that could maximize the program performance within an expected error boundary, by randomly choosing a combination of function implementations. These models and algorithms have built a meaningful foundation of approximated computation and how to adjust the program dynamically. For example, [10] suggests a way to transform the static configuration into dynamic variables. The system monitors the program behavior within a heartbeat framework and adjust those parameters when it observes that the performance has reached the fixed threshold. However, users are still responsible for providing the set of parameters to let the system initialize the trade-off space, and the quality of a program(service) is restricted as the performance. And it is far from enough in a real environment, which requires a much more complicated definition of quality. Moreover, these systems ignores the dependency between several services working together as a whole system. RSDG, in contrast, intends to select the optimal execution path among several combinations of both implementations and different approximation levels of different services. It could be applied on more general environments, rather than only a single program's execution.

Proving acceptability of approximated computation: Researchers have also developed techniques to reason about approximate computations and to verify whether adjusting the program execution could still keep the program's intended semantics. [5] suggests a sound verification method by using Coq. It differs from other verifications by trying to find the relationship between both the original (most expensive execution) and the relaxed (approximated) executions. And [6] points out a way to check that a program's output is within a boundary when adjusting the input. In RSDG, adjusting the input means not only the different input parameters of a service, but also selecting a lower level implementation at the source end of a dependence edge. So having a way to check the error bound is crucial.

Energy consumption in smartphones: There have been research on maximizing the battery life in cell phones. The author of [1]has pointed out a way to minimize the localization cost, based on the fact that if an object is detected to be stationary by a low cost device like accelerometer, there will be no need to keep consuming power by requiring GPS data. And [23] pointed out another approach, dynamically switch between GPS and network-based localization service by letting machines be smart enough to choose the cheaper implementation within precision constraints when users enter a known location, after monitoring the performance of GPS/networked-based localization service. Also, there have been hardware approaches, like [18], from the view of cache design, and [16], a controller that could improve the battery life by optimizing the duration of current. So all these approaches aim the same goal, maximizing the battery life, by optimizing a single service in the machine. Neither of them talked about a system that could describe the relations between services. But RSDG could efficiently achieve this goal, by setting up the service dependencies. For example, LCD is like the navigation system in gliders, most but all services rely on LCD to be meaningful to users. Therefore there would be edges from backlight to services like view management service. And cellular network should be turned on constantly and there would be one node in localization services depend on cellular network. When users or applications inform the system that a set of services with at least certain levels is required, e.g. manually setting up brightness by users, or location required by applications, RSDG would translate these requirements to variable constraints and come up with a solution, which is a selection of services that requires the least energy.

6. Conclusion and Future Work

The AIR framework allows users to express and optimize different dimensions of redundancies that exist in service-based applications (approximation, implementation, and replication). The redundant services dependence graph (RSDG) is a dual-weighted graph where nodes are associated with services, and edges model service dependencies. Node and edgeweights model energy requirements. Nodes may have a second weight, called the mission value, to mark them as critical for the application, and to rank their relative importance. The optimal RSDG node selection problem picks the critical nodes and all nodes it dependents on such that the overall mission value is maximized under a user defined energy budget. This problem is shown to be NP-complete. AIR includes a compiler that translates instances of the optimal RSDG node selection problem into equivalent integer programming problems that can be solved by mixed integer programming tools. Experimental results using a realistic example of a service-based system architecture, namely the mission management of an autonomous underwater vehicle (Slocum glider), show that the AIR framework and its RSDG representation is expressive and efficient. The resulting optimal solutions are often non-intuitive, motivating a need for a tool such as AIR to manage redundancies. We also investigate the AIR framework in the context of single service/component failures. By recomputing the optimal solution without considering the failed

service, a system reconfiguration was often able to provide a similar level of overall system quality by exploiting different dimensions of redundancies. Reconfiguration can also be performed dynamically, in response to changing energy resources or failure conditions. AIR also enables in-depth analyses of producivity and resilience properties of applications. We are planning to apply our AIR framework to other service-based applications such as smart phones, and perform additional field experiments with our Slocum glider. Improvements to the AIR user interface are currently underway.

References

[1] F.Ben Abdesslem, A. Phillips, and T.Henderson. Less is more: energy-efficient mobile sensing with senseless. In *1st ACM workshop on Networking, systems, and applications for mobile handhelds, MobiHeld*, pages 61–62, 2009.

[2] Alberto Alvarez, R. Stoner, and A. Maguer. Performance of pumped and un-pumped CTDs in an underwater glider. In *OCEANS 2013 IEEE - San Diego*, pages 1–5, 2013.

[3] R. Bixby. Implementing the Simplex method: The initial basis. *ORSA Journal on Computing*, 4(3):267–284, 1992.

[4] R. Bixby, K. Kennedy, and U. Kremer. Automatic data layout using 0-1 integer programming. In *Conference on Parallel Architectures and Compilation Techniques, PACT '94, Montreal, Canada, 24–26 August, 1994*, volume A-50, Amsterdam, The Netherlands, 1994. North-Holland Publishing Co.

[5] Michael Carbin, Deokhwan Kim, Sasa Misailovic, and Martin C. Rinard. Proving acceptability properties of relaxed nondeterministic approximate programs. In *PLDI'12*, Beijing, China, June 2012.

[6] Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara Navidpour. Proving programs robust. November 2011.

[7] Charles C. Eriksen, T. James Osse, Russell D. Light, Timothy Wen, Thomas W. Lehman, Peter L. Sabin, John W. Ballard, and Andrew M. Chiodi. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. In *IEEE Journal of Oceanic Engineering*, volume 26, October 2001.

[8] Scott Glenn, Oscar Schofield, Josh Kohut, Janice McDonnell, Richard Ludescherand Dena Seidel, Dadid Aragon, Tina Haskins, Ethan Handel, Clinton Haldeman, Igor Heifetz, John Kerfoot, Erick Lemus, Sage Lictenwalner, Lisa Ojanen, Jugh Roarty, Filipa Carvalho, Alvaro Lopez, Adri Martin, Clayton Jones, Douglass Webb, Jerry Miller, Marlon Lewis, Scott McLean, Ana Martins, Carlos Barrera, Antonio Ramos, and Enrique Fanjul. The trans-atlantic Slocum glider expeditions: A catalyst for undergraduate participation in ocean science and technology. *Marine Technology Society Journal*, 45(1):52–67, January/February 2011.

[9] Brett W. Hobson, James G. Bellingham, Brian Kieft, Rob McEwen, Michael Godin, and Yanwu Zhang. Tethys-class long range AUVs - extending the endurance of propeller-driven cruising AUVs from days to weeks. In *Proceedings of IEEE-OES AUV Symposium*. Southampton, U.K., September 2012.

[10] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS'11*, Newport Beach, California, USA, March 2011.

[11] Persistor Instruments Inc. Cf1 computer system. Marstons Mills, MA. http://www.persistor.com.

[12] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *PLDI'11*, San Jose, California, USA, June 2011.

[13] Oscar Schofield, Josh Kohut, David Aragon, Liz Creed, Josh Graver, Chip Haldeman, John Kerfoot, Hugh Roarty, Clayton Jones, Doug Webb, and Scott Glenn. Slocum gliders: Robust and ready. *Journal of Field Robotics*, 24(6):473–485, 2007.

[14] Jeff Sherman, Russ E. Davis, W.B. Owens, and J. Valdes. The autonomous underwater glider Spray. In *IEEE Journal of Oceanic Engineering*, volume 26, October 2001.

[15] Stelios Sidiroglou, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *ESEC/FSE'11*, Szeged, Hungary, September 2011.

[16] T. Srivastava and A. Narayan. Dynamic fuzzy controller based multilayered architecture for extended battery life in mobile handheld devices. In *Systems, Man and Cybernetics, SMC*, pages 3035–3040, 2009.

[17] Matthew D. Steele. *A Programming Language for Precision/Cost Tradeoffs*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology., May 2009.

[18] C.-L. Su and A. M. Despain. Cache design tradeoffs for power and performance optimization: a case study. In *1995 international symposium on Low power design, ISLPED*, pages 63–68, 1995.

[19] Teledyne Webb Research. Slocum Glider. Falmouth, MA, 2013. http//www.webbresearch.com/slocum.htm.

[20] The Gurobi Optimizer 6.0. Gurobi GmbH. Bad Homburg, Germany. http://www.gurobi.com.

[21] Mixed Integer Programming Tool. lpsolve. //http://lpsolve.sourceforge.net/5.5/.

[22] Zeyuan Allen Zhu, Sasa Misailovic, Jonathan A. Kelner, and Martin Rinard. Randomized accuracy-aware program transformations for efficient approximate computations. In *POPL'12*, Philadelphia, PA, USA, January 2012.

[23] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *8th international conference on Mobile systems, applications, and services, MobiSys*, pages 315–330, 2010.