Intelligent Resource Management and Dynamic Adaptation in a Distributed Real-time and Embedded Sensor Web System

John S. Kinnebrew, William R. Otte, Nishanth Shankaran,
Gautam Biswas, and Douglas C. Schmidt
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37203, USA

***Abstract*—Sensor webs are often composed of servers connected to distributed real-time embedded (DRE) systems that operate in open environments where operating conditions, workload, resource availability, and connectivity cannot be accurately characterized a priori. The South East Alaska MOnitoring Network for Science, Telecommunications, Education, and Research (SEAMONSTER) project exhibits many common system management and dynamic operation challenges for effective, autonomous system adaptation in a representative sensor web. These challenges cover both field operation (*e.g.*, power management through system sleep/wake cycles and reaction to local environmental changes) and server operation (*e.g.*, system adaptation for new/modified goals, resource allocation for a changing set of applications, and configuration changes for fluctuating workload). This paper presents the results of integrating and applying quality-of-service (QoS)-enabled component middleware, dynamic resource management, and autonomous agent technologies to address these challenges in SEAMONSTER.**

I. Introduction

The advent of sensor webs [1] help scientists study and predict weather, natural disasters, and climate change. Sensor webs are large-scale systems including both commodity servers and distributed real-time embedded (DRE) systems with several interacting subsystems that enable scientific study of environmental processes, such as weather monitoring/forecasting, ecosystem monitoring, and monitoring of earth's geological activities, in real-time. Effective sensor webs also facilitate the real-time analysis and recovery of large volumes of collected scientific data.

Similar to other DRE systems, such as such as shipboard computing [2] and fractionated spacecraft [3], sensor webs must perform sequences of heterogeneous data collection, manipulation, and coordination tasks to meet specified system objectives. Moreover, to use their limited resources effectively, sensor webs must adapt their operation to changing conditions and objectives.

The use of *quality-of-service (QoS)-enabled component middleware* in DRE systems is gaining momentum since it automates remoting, lifecycle management, system resource management, deployment, and configuration in these systems. QoS-enabled component middleware support explicit configuration of QoS aspects (*e.g.*, priority and threading models), and provide many desirable real-time features (*e.g.*, priority propagation, scheduling services, and explicit binding of network connections). In integrated, adaptive sensor webs, QoS-enabled component middleware helps address the larger set of assets and computational resources that must be coordinated and managed to address weather, climate change, and disaster management problems.

Sensor web hardware and sensors are also increasingly configurable and must operate in *open* environments where operating conditions, workload, resource availability, and connectivity cannot be accurately characterized *a priori*. The challenges presented by such open environments are only recently being addressed in DRE systems [4]. The combination of QoS-enabled component middleware with dynamic resource allocation/control and agents for intelligent, local autonomy in achieving science objectives provides a powerful solution approach to many system management and dynamic operation challenges facing sensor webs.

This paper presents a case study where a combination of middleware, resource management, and autonomous agent technologies are applied to the *South East Alaska MOnitoring Network for Science, Telecommunications, Education, and Research* (SEAMONSTER) [5], which is a representative sensor web for monitoring glacial change and watershed effects. System adaptation and management challenges for effective use of limited resources in SEAMONSTER are presented, including field operation challenges (*e.g.*, power management through system sleep/wake cycles and goal-driven reaction to local environmental changes) and server operation challenges (*e.g.*, system adaptation for new/modified goals, resource allocation for a changing set of applications, and configuration changes for fluctuating workload).

We then describe how we have addressed the SEAMONSTER challenges as part of the development of the *Multi-agent Architecture for Coordinated Responsive Observations* (MACRO) [6] platform. MACRO uses QoS-enabled component middleware to help automate many system configuration and management tasks for sensor web applications. Atop the middleware infrastructure, MACRO's dynamic resource management services provide efficient allocation and control of computational resources, while MACRO agents employ reasoning and planning to autonomously adapt system functionality to changing science objectives and environmental conditions. We conclude by summarizing key lessons learned from our application of the MACRO platform to the SEAMONSTER hardware.

II. Sensor Web Case Study: SEAMONSTER

A. Overview of SEAMONSTER

SEAMONSTER is a glacier and watershed sensor web in Alaska [5]. This sensor web monitors and collects data regarding glacier dynamics and mass balance, watershed hydrology, coastal marine ecology, and human impact/hazards in and around the Lemon Creek watershed. The collected data is used to study the correlation between hydrology, glacier velocity, and temperature variation at Lemon Creek.

The SEAMONSTER sensor web, as illustrated in Figure 1, includes sensors with weatherized computer platforms that are deployed on the glacier and throughout the watershed to collect data of scientific interest. The data collected by the sensors is relayed to a cluster of servers primarily via wireless networks for processing, correlation, and analysis. These data processing applications are being transitioned to run atop a QoS-enabled component middleware platform consisting of the *Component-Integrated ACE ORB* (CIAO) [7], which is open-source QoS-enabled component middleware that implements the OMG Lightweight CORBA Component Model (CCM) [8] and Deployment and Configuration [9] specifications.

The data processing applications (*e.g.*, GPS data analysis for glacier dynamics and watershed hydrology analysis) are configured by autonomous sensor web agents on the server cluster, and individual applications may be added or removed to/from the server cluster during normal operation. The resource utilization by these applications cannot be accurately characterized *a priori* since it depends on the input workload for these applications, which in turn is affected by a plethora of environmental conditions and activities in the field.

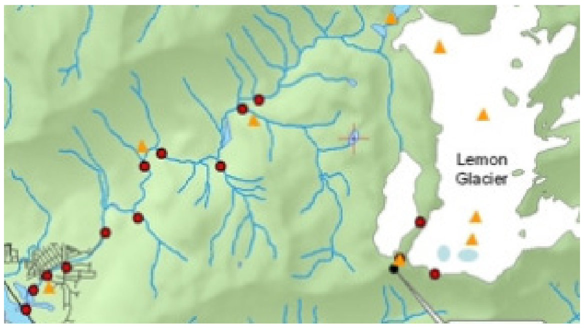


Fig. 1: SEAMONSTER field sensor deployment

For example, during nominal operation of the SEAMONSTER sensor web, only a subset of the sensors are operational (primarily for baseline monitoring of the Lemon Creek Glacier and Lemon Creek watershed area). The input workload of the applications processing the collected data is therefore minimal. When evidence is detected that the glacial lake on Lemon Creek Glacier is draining, however, most or all of the sensors in the sensor web should be operational and configured to higher data rates, resulting in much larger quantities of sensor data being collected to allow in-depth analysis of the effects of the lake draining through the glacier into Lemon Creek. During this event, input workload of the data processing applications are significantly higher than during normal operation.

B. System Adaptation Challenges in SEAMONSTER

Effective sensor web resource management and dynamic adaptation in SEAMONSTER presents software challenges in the field and at the servers. In the server cluster, significant computational resources are available to direct the tasks performed by computationally limited field resources. These servers are shared among the data processing applications, sensor web agents, and other SEAMONSTER applications, such as a database and web server. In the field, computational resources are limited and consequently require software solutions with small footprint and low computational complexity. The remainder of this section summarizes key field and server challenges in transforming SEAMONSTER to use its limited resources effectively by applying component middleware, dynamic resource management, and autonomous agents.

1) Field Challenge 1: Local power management with sleep/wake cycles: SEAMONSTER's need for power management is motivated by limited availability of power, due to variable weather conditions limiting the ability to recharge the batteries. The available power is often insufficient for continuous operation of the processor, requiring the system to periodically power down completely. Moreover, to protect against "wedging" (which is a situation where the operating system becomes unresponsive), it is useful to periodically hard-reset the microservers, which are difficult to physically access in the field. When a microserver returns from one of these sleep/wake cycles, *i.e.* when the boot process completes, local agents and applications must be correctly re-deployed and connections between nodes must be correctly re-established. Section III-A describes how MACRO QoS-enabled component middleware addresses this challenge.

2) Field Challenge 2: Configuration adaptation to address changing local conditions: Field nodes in a sensor web often have a large number of observable phenomena in their area of interest. The type, duration, and frequency of observation of these phenomena may change over time, based on changes in the environment, occurrence of events in the environment, and changing goals and objectives in the science mission of the sensor web. Moreover, limited power, processing capability, storage, and network bandwidth limit the ability of these nodes to continually perform observations at the desired frequency and fidelity. Dynamic changes in environmental conditions coupled with limited resource availability requires individual nodes of the sensor web to revise current operations and future plans to make the best use of their resources. To handle these dynamic changes effectively, the nodes must be capable of goal-driven, functional adaptation. Section III-B describes how MACRO field agents address this challenge.

3) Server Challenge 1: Autonomous system adaptation for new or modified objectives: The scientific inquiries and corresponding mission objectives for which a sensor web is utilized can change over the course of its operation. A system with static data collection and processing activities requires significant effort to reconfigure for new or modified mission objectives. Moreover, the system may have more objectives than can all be achieved at the same time with its limited resources. The "best" (*i.e.*, highest utility) set of goals to achieve at a particular time may vary depending on environmental conditions and transient events. Efficient use of sensor web resources therefore requires autonomous adaptation of system configuration and activities in light of current goals and conditions. Section III-C describes how MACRO server agents address this challenge.

4) Server Challenge 2: Online resource allocation to data processing applications: Data processing applications executing in the server cluster are *resource sensitive*, *i.e.*, QoS of the sensor web is affected significantly if an application does not receive the required CPU time and network bandwidth within bounded delay. Moreover, in open DRE systems like the SEAMONSTER sensor web, input workload affects utilization of system resources and QoS of applications. Utilization of system resources and QoS of applications may therefore vary significantly from their estimated values. Section III-D describes how MACRO resource management services address this challenge.

5) Server Challenge 3: Adaptation to fluctuations in input workload: When applications are deployed and configured in the server cluster, resources are allocated to application components based on the *estimated* resource utilization and estimated/current availability of system resources. In open DRE systems, however, *actual* resource utilization of applications may differ significantly from the estimated values. Moreover, for applications executing in these systems, the relation between input workload, resource utilization, and QoS cannot be fully characterized *a priori*. To operate effectively despite dynamic variations in operational conditions and/or input workload, the system should be able to gracefully *adapt* resource usage. Section III-E describes how MACRO resource management services address this challenge.

III. Addressing the SEAMONSTER Challenges with the MACRO Platform

The *Multi-agent Architecture for Coordinated, Responsive Observations* (MACRO) platform provides a powerful computational infrastructure for enabling the deployment, configuration, and operation of large-scale sensor webs that are composed of many constituent sensor webs. Intelligent autonomy in MACRO is provided primarily through two levels of agents: (1) the *mission level*, where agents interact with users to allocate high-level science tasks to sensor webs and create scheduled plans to achieve these goals, and (2) *resource level*, where local server and field agents translate tasks into actions and application deployments related to data collection, analysis, and transmission.

To effectively adapt system functionality, resource-level agents in MACRO employ novel services, such as the *Spreading Activation Partial Order Planner* (SA-POP) [10] and the *Resource Allocation and Control Engine* (RACE) [11]. MACRO server agents use SA-POP to support dynamic (re)planning/scheduling and RACE to efficiently manage computational resource for deployed applications. SA-POP and RACE enable MACRO to achieve the necessary local autonomy to efficiently achieve mission goals with limited resources in a dynamic environment.

The implementation of agents in MACRO is based on the CIAO [7] QoS-enabled component middleware to ensure interoperability across heterogeneous computing platforms, reduce development costs, and improve overall robustness and scalability. The agents operate on the CIAO middleware to ensure that a diverse set of science objectives can be met, as shown in Figure 2. This architecture helps facilitate real-time, adaptive data acquisition, analysis, fusion, and distribution.

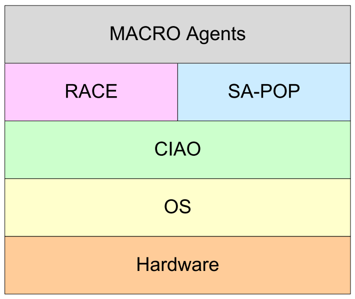


Fig. 2: **M**ACRO Architecture

The remainder of this section describes how MACRO addresses the sensor web challenges identified in Section II-B.

A. Addressing Field Challenge 1: Correct Re-deployment After Reboot

The MACRO approach to resolving this challenge involves creating all deployments as locality-constrained deployments. Locality-constrained deployments describe only components that reside on a single node and describe connections with components on other nodes with external references. The locality-constrained approach is in contrast to the use of a global deployment plan, which describes components deployed to several nodes and describes connections as internal references, *i.e.*, referring to the connected components directly. With locality-constrained deployments each node has sufficient information for the middleware to correctly reconstitute its agent and other software deployments upon reboot.

B. Addressing Field Challenge 2: Plan Schemas for Local Reaction

To address the problem of effective reaction to local changes in environmental conditions and resource availability—while respecting system-wide science goals—the field nodes must be capable of intelligent, autonomous adaptation and action. Since local field agents have limited computational resources, extensive planning and scheduling is not possible for rapid reaction to local changes. Instead, field agents are provided with a set of template plan schemas that cover a range of conditions and local goals to which they are applicable. Server-based agents can then provide the field agents with the current set of local goals to pursue, and the task of the field agent becomes the simpler choice of an appropriate set of schemas to follow given current conditions. MACRO's field agents therefore select and employ their schemas to resolve the challenge of configuration adaptation to address changing local conditions identified in Section II-B2.

C. Addressing Server Challenge 1: Planning and Schema Production to Achieve Science Objectives

System adaptation for new or modified goals, described as a set of desired data products and results, is controlled by

MACRO server-based agents with functional knowledge of the sensor web system and available software components. These agents use SA-POP to decompose goals into subgoals to be achieved at the server or by individual field nodes and to plan/schedule for their achievement. With information from field agents about current conditions and local activities, SA-POP produces scheduled, high expected utility plans to achieve all subgoals, including both the selection/configuration of software components for data processing on the server and actions/reconfiguration at the field nodes.

If the planned set of actions and software deployments at a field node is already represented in a schema available to the field node's agent, the subgoal and scheduling information is simply passed to that field agent. Otherwise, the plan is packaged as a new schema and distributed to the field agent along with the applicable subgoal and scheduling information. This process allows the server-based agents to do extensive planning and optimization for the current set of science objectives, as well as entirely new objectives, while the computationally limited field agents can choose among their pre-packaged schemas to intelligently react to changing local conditions and resource availability in light of current science objectives. MACRO's server-based agents and SA-POP can therefore resolve the challenge of intelligent system adaptation for new/modified objectives identified in Section II-B3.

D. Addressing Server Challenge 2: Online Allocation of Computational Resources

As shown in Figure 3, RACE parses the metadata that describes the application to obtain the resource requirement(s) of components that make up the application. The `Resource`

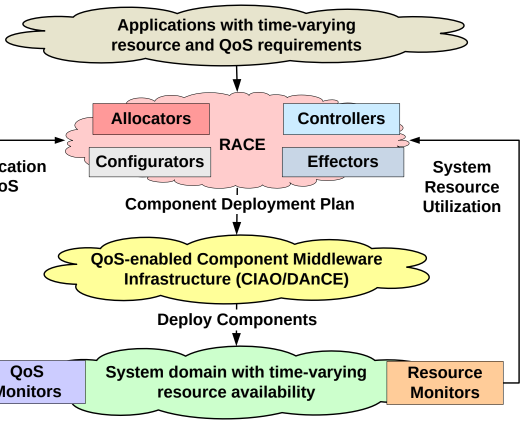

Applications with time-varying resource and QoS requirements
Allocators
Controllers
RACE
Configurators
Effectors
Application QoS
System Resource Utilization
Component Deployment Plan
QoS-enabled Component Middleware Infrastructure (CIAO/DAnCE)
Deploy Components
QoS Monitors
System domain with time-varying resource availability
Resource Monitors

Fig. 3: The RACE Architecture

`Monitor` obtains system resource utilization/availability information, and using this information along with the *estimated* resource requirement of application components captured in application's metadata, the `Allocators` (which implement resource allocation algorithms, such as single dimension bin-packing [12] and availability and partitioned breadth first decreasing [13]) map components onto nodes in the system domain based on runtime resource availability. RACE's `Resource Monitor` and `Allocators` coordinate with one another to allocate resources to applications executing in the SEAMONSTER system, thereby addressing the resource allocation requirement identified in Section II-B4.

E. Addressing Server Challenge 3: Runtime System Adaptation for Resource Management

RACE's `Allocators` allocate resources to applications based on current system resource utilization and application's estimated resource requirements. In open DRE systems, however, there is often no accurate *a priori* knowledge of input workload and the relationship between input workload and resource requirements of an application. To address this requirement, RACE's control architecture employs a feedback loop shown in Figure 4 to manage system resource and application QoS and ensures (1) QoS requirements of applications are met at all times and (2) system stability by maintaining utilization of system resources below their specified utilization set-points. RACE's control architecture features a feedback loop that consists of three main components: `Monitors`, `Controllers`, and `Effectors`.

`Monitors` are associated with system resources and QoS of the applications and periodically update the `Controller` with the current resource utilization and QoS of applications currently running in the system. The `Controller` implements a particular control algorithm such as EUCON [14], DEUCON [15], HySUCON [16], or FMUF [17], and computes the adaptation decisions for each (or a set of) application(s) to achieve the desired system resource utilization and QoS. `Effectors` modify system parameters, which include resource allocation to components, execution rates of applications, and OS/middleware/network QoS setting of components, to achieve the controller recommended adaptation.

RACE's monitoring framework, `Controllers`, and `Effectors` coordinate with one another and the aforementioned entities of RACE to ensure (1) QoS requirements of applications are met and (2) utilization of system resources are maintained within the specified utilization set-point set-point(s), thereby addressing the requirements associated with runtime end-to-end QoS management identified in Section II-B5. We empirically validate this capability as it relates to the other server challenges of SEAMONSTER in Section IV.

IV. Experimental Results

This section presents the design and results of experiments that evaluate the adaptive resource management capabilities of RACE in the context of the SEAMONSTER servers and agents described in Section II-A. These experiments also validate our claims in Section III that RACE performs effective end-to-end adaptation and yields a predictable and scalable DRE system under varying operating conditions and input workload.

A. Hardware and Software Testbed

Our experiments were performed on the ISISLab testbed (www.dre.vanderbilt.edu/ISISlab) at Vanderbilt University, which is a cluster consisting of 56 IBM blades powered by Emulab software (www.emulab.net). Each blade node contains

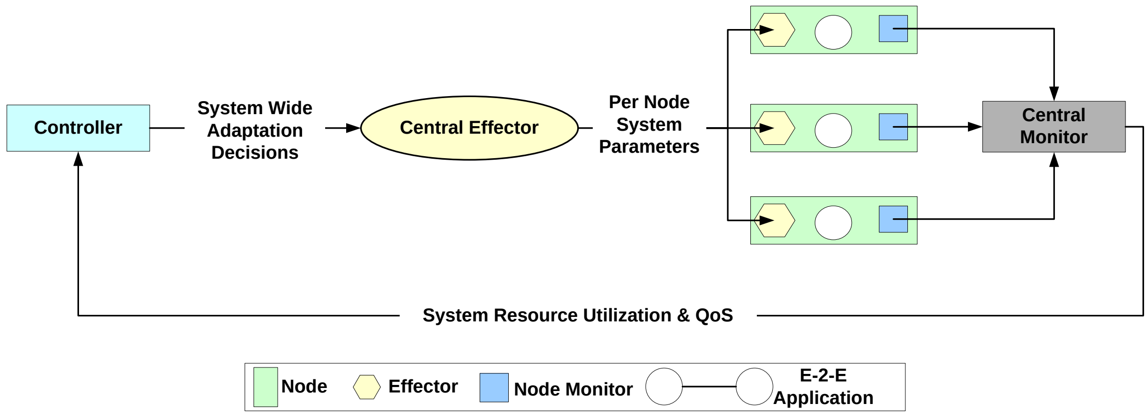

Controller
System Wide Adaptation Decisions
Central Effector
Per Node System Parameters
Central Monitor
System Resource Utilization & QoS
Node
Effector
Node Monitor
E-2-E Application

Fig. 4: RACE's Feedback Control Loop

two 2.8 GHz Intel Xeon processors, 1 GB physical memory, 1GHz Ethernet network interface, and 40 GB hard drive. The Redhat Fedora Core release 4 OS with real-time preemption patches [18] was used on all nodes. We used five blade nodes for the experiments to emulate the server cluster of our prototype SEAMONSTER sensor web. Our middleware platform was CIAO version 0.5.10.

B. System Implementation and Experiment Design

The set of components making up data processing applications on our SEAMONSTER sensor web testbed are configured by the server-based MACRO agents and can be classified as (1) glacier dynamics monitoring, (2) watershed hydrology analysis, and (3) coastal marine ecology analysis applications. These applications were periodic (*i.e.*, applications contained a timer component that periodically triggered the collection from simulated field nodes, followed by the data filtering and analysis on the server) and the execution rate of these applications could be modified at runtime.

As described in section II-B, the SEAMONSTER sensor web is subject to fluctuations in application workload. To validate our claim that RACE enables the autonomous operation of open DRE systems, such as the SEAMONSTER sensor web, by providing effective end-to-end resource management adaptation to MACRO agents, we evaluated performance of our prototype SEAMONSTER sensor web performance when application workloads were varied at runtime. Our experiment compares the performance of the system that is subjected to fluctuations input workload when the system is operated with and without RACE. As execution rates of applications that executed in this system could be dynamically modified at runtime, RACE was configured to employ the EUCON [14] control algorithm to compute system adaptation decisions.

C. Evaluation of RACE's Adaptive Resource Management Capabilities

This experiment varied the input workload to data processing applications at runtime to demonstrate the adaptive resource management capabilities of RACE under varying input workload by comparing system performance with and without RACE. We use deadline miss ratio, average application throughput, and system resource utilization as metrics to empirically compare the performance of the system under each service configuration.

#	exec. rate (hz)			estimated	component average resource util.				
	min	max	init.	util.	1	2	3	4	5
1	15	155	60	0.3	0.15	0.1	0.05	0	0
2	35	165	85	0.1	0.05	0.05	0	0	0
3	10	140	50	0.5	0.2	0.1	0.1	0.05	0.05
4	30	170	80	0.3	0.25	0.05	0	0	0
5	35	180	90	0.45	0.2	0.1	0.1	0.05	0
6	10	140	65	0.35	0.15	0.1	0.05	0.05	0
7	35	170	95	0.35	0.25	0.05	0.05	0	0

TABLE I: Application Configuration

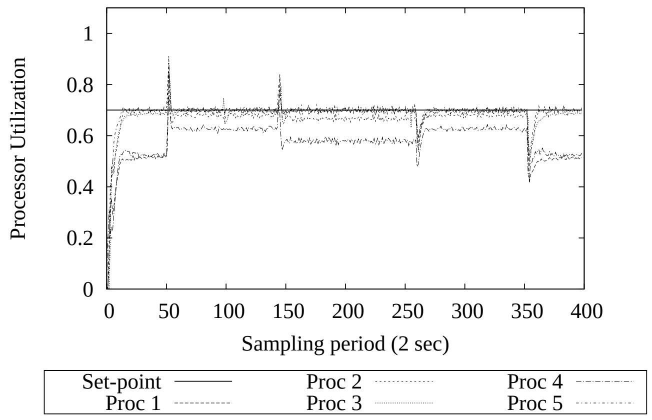
1) Experiment Configuration: At time $t = 0$, the system was initialized with the applications specified in table I to perform glacier dynamics monitoring, watershed hydrology analysis, and coastal marine ecology analysis. Upon initialization, applications execute at their initialization rate specified in table I. Each applications end-to-end deadline is defined as $d_i = n_i/r_i(k)$, where n_i is the number of components in application t_i and $r_i(k)$ is the execution rate of application t_i in the k^{th} sampling period. Each end-to-end deadline is evenly divided into sub-deadlines for its components. The resultant sub-deadline of each component equals its period, $1/r(k)$. All applications/components meet their deadlines/sub-deadlines if the schedulable utilization bound of rate monotonic scheduling (RMS) [12] is used as the utilization set-point and is enforced on all the nodes.

The sampling period of the controller was set at 2 seconds and the utilization set-point for each node was selected to be 0.7, which is slightly lower than the RMS utilization bound. Table II summarizes the variation of input workload as a function of time. When the input workload was low, medium, and high, the corresponding resource utilization by application components were their corresponding best case, average case, and worst case values, respectively.

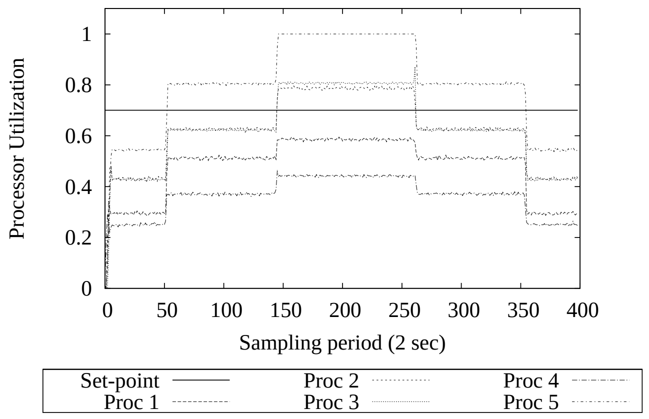
2) Analysis of Experiment Results: When RACE is available as a service to MACRO agents, it dynamically modifies the execution rates of applications within the bounds $[min, max]$ specified in table I to ensure that the resource utilization on each node converges to the specified set-point

sampling period	input workload
0 - 50	low
50 - 150	medium
150 - 250	high
250 - 350	medium
350 - 400	low

TABLE II: Input Workload as a Function of Time



(a) With RACE



(b) Without RACE

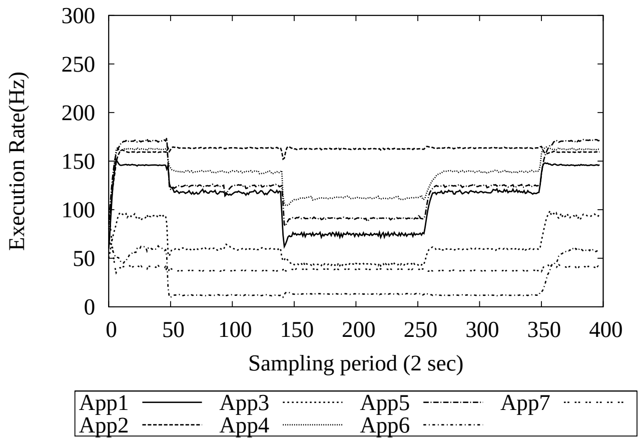
Fig. 5: Comparison of Processor Utilizations

of 0.7, despite fluctuations in input workload. When the system operated without RACE, however, applications executed at their initialization rates set by MACRO agents for expected workload conditions and identified in table I.

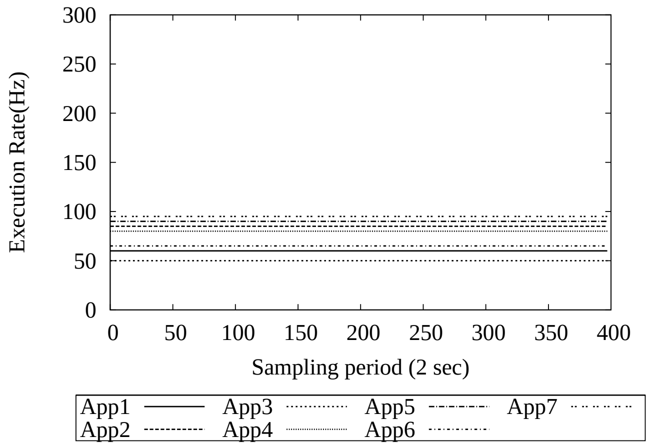
Figure 5a, Figure 6a, and Table II show the execution of the system when RACE is employed. During $0 \leq t \leq 100$, when the input workload is low, the controller increases the execution rates of applications such that the processor utilization on each node converges to the desired set-point of 0.7. This behavior ensures effective utilization of system resources. When RACE is not used, however, figures 5b and 6b show that the applications must execute at a constant rate (initialization rate) and system resources are severely underutilized.

When input workload is increased from low to medium, at $t = 100s$, the corresponding increase in the processor utilization can be seen in figure 5. Figures 5a and 6a show that when RACE is used, although the processor utilization increased above the set-point, within a few sampling periods the controller restored the processor utilization to the desired set-point of 0.7 by dynamically reducing the execution rates of applications. The deadline miss ratio for the entire duration of the experiment was observed to be 0.005 and 0.0184 when the system was operated with and without RACE, respectively. Figure 5b shows that without RACE, the processor utilization was below the set-point for all the nodes in the system, except for node 5.

At $t = 300s$, the input workload was further increased from medium to high. As a result, the processor utilization on all the nodes increased, which is shown in figure 5. Figures 5a and 6b show that RACE was again able to dynamically modify the application execution rates to ensure that the utilization converged to the desired set-point. Figure 5b shows that without RACE, the processor utilization on most of the nodes in the system was significantly higher than the set-point under high workload conditions.



(a) With RACE



(b) Without RACE

Fig. 6: Comparison of Application Execution Rates

At $t = 500s$, when the input workload was reduced from high to medium, from figure 5 it can be seen that the processor utilization on all the nodes decreased. When the system was operated with RACE, however, RACE restored the processor utilization to the desired set-point of 0.7 within a few sampling periods. Without RACE, processor utilization for all nodes except node 5 remained significantly lower than the set-point. Similarly, at $t = 700s$, the input workload was further reduced from medium to low, and figure 5 shows another decrease in processor utilization across all nodes. When the system featured RACE, processor utilization again returned to the desired set-point within a few sampling periods. Without RACE, processor utilization remained even further below the set-point.

Figure 5 shows that system resources may be either significantly underutilized or over-utilized when operating without RACE, but are near the set-point when RACE is used. Under-utilization and/or over-utilization of system resources results in reduced QoS, which is evident from table III, showing the overall system QoS.[1] In contrast, when the system featured RACE, the application execution rates were dynamically modified to ensure utilization on all the nodes converged to the set-point, resulting in more effective utilization of system resources and higher QoS.

application	Average Throughput (hz)	
	With RACE	Without RACE
1	110.326	59.930
2	160.891	84.903
3	60.532	45.964
4	133.894	76.909
5	124.232	89.599
6	21.476	63.2362
7	37.264	94.896
entire system	92.660	74.445

TABLE III: Comparison of System QoS

3) Summary: This experiment compared system performance during input workload fluctuations when the system was operated with and without RACE. The results show how RACE (1) ensures system resources are not over-utilized, (2) improves overall system QoS, and (3) enables the system to adapt to drifts/fluctuations in utilization of system resources by *fine-tuning* application parameters.

V. Related Work

Compared to related research presented in [19], the resource management framework used in MACRO – RACE – is an adaptive resource management framework that can be customized and configured using model-driven deployment and configuration tools such as the *Platform-Independent Component Modeling Language* (PICML) [20]. Moreover, RACE provides adaptive resource and QoS management capabilities more transparently and non-intrusively than Kokyu [21], QuO [22], and Qoskets [23], [24], [25]. In particular, it allocates CPU, memory, and networking resources to application components and tracks and manages utilization of various system resources, as well as application QoS.

The planning service used by MACRO server-based agents – SA-POP – is a decision-theoretic planner allowing uncertainty both in environmental conditions and action outcome, like C-SHOP [26] that does so with hierarchical planning and Drips [27] that produces conditional plans. However, to enable planning with resource constraints, such as those of sensor webs, many have chosen to separate the planning and scheduling/resource aspects of the problem. This approach works well when the resource/time constraints are relatively loose or there are relatively few alternatives in the planning process that could use fewer or different resources. However, with tight resource constraints, as are often present in sensor webs, others have chosen to integrate planning and scheduling as SA-POP does. For example, IxTeT [28] uses partial-order planning like SA-POP and allows interleaving resource conflict resolution with the planning process, but does not perform decision-theoretic planning.

The MACRO field agents use plan schemas (also called template plans or skeletal plans) [29] as have been used in other situations where complete planning was too time consuming for appropriate responses. Plan schemas have also been enhanced with scheduling information, such as in [30], and generated through partial order planning techniques, like [31]. However the combination of MACRO server-based agents using the SA-POP planning/scheduling service with generated schemas used by MACRO field agents provides a uniquely flexible solution for autonomy in sensor webs with a server cluster connected to DRE field systems.

VI. Concluding Remarks

This paper presents the results of integrating and applying the MACRO platform to address these challenges in the SEAMONSTER sensor web. The lessons learned from our activities thus far include:

- **Maintaining QoS in open DRE systems.** To make the best use of limited resources and achieve high end-to-end QoS in open DRE systems requires goal-driven functional adaptation of system configuration, adaptive resource management, and an infrastructure providing a range of real-time capabilities. QoS-enabled component middleware, such as CIAO, provide an integrated platform for building these systems and are emerging as proven operating platform for these systems. Although CIAO alleviates many challenges in building DRE systems, it does not addresses the adaptive resource management challenges and requirements of open DRE systems. Adaptive resource management solutions, such as RACE, are therefore needed to ensure QoS requirements of applications executing atop these systems are met. Moreover, goal-driven, functional system adaptation in light of current conditions, such as by autonomous MACRO agents using SA-POP and plan schemas, is required to maximize system utility and application QoS with limited sensor web resources.
- **Agents with different levels of planning/scheduling capabilities.** The extremely limited computational hardware on which SEAMONSTER field agents execute prevents them from effectively using advanced planning and scheduling techniques. A key requirement for dynamic sensor web operation, however, is rapid reaction to changing environmental conditions based on functional system knowledge and science objectives. A simple two-level hierarchy of agents

[1]In the SEAMONSTER system, overall QoS is defined as the total throughput for all active applications.

with differing capabilities can address these issues in sensor webs such as SEAMONSTER. In the field, the use of agents with simple plan schemas allows the necessary autonomy and rapid local reaction, while production of the applicable schemas by server-based agents with system-wide knowledge and advanced planning/scheduling capabilities allows dynamic adaptation to changing mission objectives.

• **Decoupling resource management algorithms from middleware and agents.** Implementing adaptive resource management algorithms within the middleware tightly couples the resource management algorithms within particular middleware platforms. This coupling makes it hard to enhance the algorithms without redeveloping significant portions of the middleware. Alternatively, direct execution and resource management of deployed data analysis applications by agents is an untenable solution for systems in which computational resources may be used by applications outside the agents' control (*e.g.*, databases and web servers that also execute on the SEAMONSTER server cluster). Adaptive resource management frameworks, such as RACE, improve flexibility by decoupling resource management algorithms from either middleware platforms or agent executors.

RACE, SA-POP, and CIAO are open-source software and can be obtained from download.dre.vanderbilt.edu.

REFERENCES

[1] K. Delin and S. Jackson, "Sensor Web for In Situ Exploration of Gaseous Biosignatures," 2000.

[2] P. Lardieri, J. Balasubramanian, D. C. Schmidt, G. Thaker, A. Gokhale, and T. Damiano, "A Multi-layered Resource Management Framework for Dynamic Resource Management in Enterprise DRE Systems," *Journal of Systems and Software: Special Issue on Dynamic Resource Management in Distributed Real-time Systems*, vol. 80, no. 7, pp. 984–996, July 2007.

[3] O. Brown and P. Eremenko, "Fractionated Space Architectures: A Vision for Responsive Space," in *Proceedings of the 4th Responsive Space Conference*. Los Angeles, CA: American Institute of Aeronautics & Astronautics, 2006.

[4] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 7, pp. 996–1009, 2007.

[5] D. R. Fatland, M. J. Heavner, E. Hood, and C. Connor, "The SEAMONSTER Sensor Web: Lessons and Opportunities after One Year," *AGU Fall Meeting Abstracts*, pp. A3+, Dec. 2007.

[6] D. Suri, A. Howell, D. C. Schmidt, G. Biswas, J. Kinnebrew, W. Otte, and N. Shankaran, "A Multi-agent Architecture for Smart Sensing in the NASA Sensor Web," in *Proceedings of the 2007 IEEE Aerospace Conference*, Big Sky, Montana, Mar. 2007.

[7] N. Wang, D. C. Schmidt, A. Gokhale, C. Rodrigues, B. Natarajan, J. P. Loyall, R. E. Schantz, and C. D. Gill, "QoS-enabled Middleware," in *Middleware for Communications*, Q. Mahmoud, Ed. New York: Wiley and Sons, 2004, pp. 131–162.

[8] *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 ed., Object Management Group, May 2003.

[9] *Deployment and Configuration Adopted Submission*, OMG Document mars/03-05-08 ed., Object Management Group, July 2003.

[10] J. S. Kinnebrew, A. Gupta, N. Shankaran, G. Biswas, and D. C. Schmidt, "Decision-Theoretic Planner with Dynamic Component Reconfiguration for Distributed Real-time Applications," in *The 8th International Symposium on Autonomous Decentralized Systems (ISADS 2007)*, Sedona, Arizona, Mar. 2007.

[11] N. Shankaran, D. C. Schmidt, Y. Chen, X. Koutsoukous, and C. Lu, "The Design and Performance of Configurable Component Middleware for End-to-End Adaptation of Distributed Real-time Embedded Systems," in *Proc. of the 10th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2007)*, Santorini Island, Greece, May 2007.

[12] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *RTSS '89: Proceedings of the IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1989, pp. 166–171.

[13] D. de Niz and R. Rajkumar, "Partitioning Bin-Packing Algorithms for Distributed Real-time Systems," *International Journal of Embedded Systems*, vol. 2, no. 3, pp. 196–208, 2006.

[14] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-time Systems with End-to-End Tasks," *IEEE Trans. on Par. and Dist. Sys.*, vol. 16, no. 6, pp. 550–561, 2005.

[15] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "Decentralized utilization control in distributed real-time systems," in *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 133–142.

[16] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu, "Hybrid Supervisory Control of Real-time Systems," in *IEEE Real-time and Embedded Technology and Applications Symposium*. San Francisco, California: IEEE Computer Society, Mar. 2005.

[17] Y. Chen and C. Lu, "Flexible Maximum Urgency First Scheduling for Distributed Real-Time Systems," Washington University in St. Louis, Tech. Rep. WUCSE-2006-55, October 2006.

[18] I. Molnar, "Linux with Real-time Pre-emption Patches," http://www.kernel.org/pub/linux/kernel/projects/rt/, Sep 2006.

[19] D. Fleeman, M. Gillen, A. Lenharth, M. Delaney, L. R. Welch, D. W. Juedes, and C. Liu, "Quality-Based Adaptive Resource Management Architecture (QARMA): A CORBA Resource Management Service," in *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. IEEE Computer Society, 2004.

[20] K. Balasubramanian, J. Balasubramanian, J. Parsons, A. Gokhale, and D. C. Schmidt, "A platform-independent component modeling language for distributed real-time and embedded systems," in *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 190–199.

[21] C. D. Gill, "Flexible Scheduling in Middleware for Distributed Rate-Based Real-time Applications," Ph.D. dissertation, Department of Computer Science, Washington University, St. Louis, 2002.

[22] J. A. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 1–20, 1997.

[23] R. Schantz, J. Loyall, M. Atighetchi, and P. Pal, "Packaging Quality of Service Control Behaviors for Reuse," in *Proceedings of the 5^{th} IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*, Crystal City, VA, April/May 2002, pp. 375–385.

[24] P. K. Sharma, J. P. Loyall, G. T. Heineman, R. E. Schantz, R. Shapiro, and G. Duzan, "Component-based dynamic qos adaptations in distributed real-time and embedded systems," in *CoopIS/DOA/ODBASE (2)*. Agia Napa, Cyprus: Springer, 2004, pp. 1208–1224.

[25] P. Manghwani, J. Loyall, P. Sharma, M. Gillen, and J. Ye, "End-to-End Quality of Service Management for Distributed Real-Time Embedded Applications," in *18th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, vol. 03, Los Alamitos, CA, USA, 2005.

[26] A. Bouguerra and L. Karlsson, "Hierarchical Task Planning Under Uncertainty," *3rd Italian Workshop on Planning and Scheduling (AI* IA 2004). Perugia, Italy*, 2004.

[27] P. Haddawy, A. Doan, and R. Goodwin, "Efficient Decision-Theoretic Planning: Techniques and Empirical Analysis," *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995.

[28] P. Laborie and M. Ghallab, "Planning with Sharable Resource Constraints," *Proc. 14th Int. Joint Conf. on AI*, pp. 1643–1649, 1995.

[29] P. Friedland and Y. Iwasaki, "The concept and implementation of skeletal plans," *Journal of Automated Reasoning*, vol. 1, no. 2, pp. 161–208, 1985.

[30] S. Miksch, Y. Shahar, and P. Johnson, "Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans," in *Proceedings of the 7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)*, 1997, pp. 9–19.

[31] L. Ihrig and S. Kambhampati, "Design and Implementation of a Replay Framework Based on a Partial Order Planner," in *Proceedings of the National Conference on Artificial Intelligence*, 1996, pp. 849–854.