An Integrated Planning and Adaptive Resource Management Architecture for Distributed Real-time Embedded Systems

Nishanth Shankaran, John S. Kinnebrew, Xenofon D. Koutsoukos
Chenyang Lu, Douglas C. Schmidt, and Gautam Biswas

Abstract—Real-time and embedded systems have traditionally been designed for closed environments where operating conditions, input workloads, and resource availability are known a priori and are subject to little or no change at runtime. There is increasing demand, however, for autonomous capabilities in open distributed real-time and embedded (DRE) systems that execute in environments where input workload and resource availability cannot be accurately characterized a priori. These systems can benefit from autonomic computing capabilities, such as self-(re)configuration and self-optimization, that enable autonomous adaptation under varying—even unpredictable—operational conditions.

A challenging problem faced by researchers and developers in enabling autonomic computing capabilities to open DRE systems involves devising adaptive planning and resource management strategies that can meet mission objectives and end-to-end quality of service (QoS) requirements of applications. To address this challenge, this paper presents the Integrated Planning, Allocation, and Control (IPAC) framework, which provides decision-theoretic planning, dynamic resource allocation, and runtime system control to provide coordinated system adaptation and enable the autonomous operation of open DRE systems.

This paper presents two contributions to research on autonomic computing for open DRE systems. First, we describe the design of IPAC and show how IPAC resolves the challenges associated with the autonomous operation of a representative open DRE system case study. Second, we empirically evaluate the planning and adaptive resource management capabilities of IPAC in the context of our case study. Our experimental results demonstrate that IPAC enables the autonomous operation of open DRE systems by performing adaptive planning and management of system resources.

1 Introduction

Many mission-critical *distributed real-time and embedded* (DRE) systems must operate in *open* environments where operating conditions, input workload, and resource availability cannot be accurately characterized *a priori*. Examples include multi-satellite systems [1] and fractionated space systems [2].

Autonomous operation of complex systems, including open DRE systems, require them to *adapt* system operation and/or functionality in response to changing mission goals and environmental conditions [3]. To autonomously adapt system functionality in this manner requires that applications be specifically tailored to current goals and conditions. Such dynamic application assembly/modification presents significant planning/re-planning challenges in DRE systems. For example, uncertainty in the outcome of actions and the limited and changing resource availability must be considered in an efficient (re)planning system, which adapts system functionality to current objectives and local conditions.

For effective autonomous operation of open DRE systems, functional adaptation alone, however, is insufficient. Achieving end-to-end quality of service (QoS) in these systems also requires the resolution of resource management challenges governed by various factors. For example, these systems often have multiple interdependent resource constraints (*e.g.*, limited computing power, storage, battery power, and network bandwidth) and highly fluctuating resource availability and input workload. They must also support simultaneous execution of multiple end-to-end applications of varying degrees of importance. In addition, application components may be added and/or removed at runtime as the result of system adaptation through planning, *e.g.*, when mission goals are added/changed, local conditions change unexpectedly, or failure and/or loss of resources occurs.

Conventional resource management approaches, such as real-time task allocation and scheduling mechanisms [4], are poorly suited to open DRE systems due to the dynamic and uncertain environments and requirements of these systems. A promising solution is *feedback control scheduling* (FCS) [5], [6], [7], which employs software feedback loops that dynamically control resource allocation to applications in response to changes in input workload and resource availability. FCS algorithms, however, have limited applicability to open DRE system that operate autonomously. Autonomous operations require systems to adapt to a *combination* of changes in mission requirements and goals, changes in operating/environmental conditions, loss of resources, and drifts or fluctuations in system resource utilization and application QoS at runtime.

Adaptation in open DRE systems can be performed at multiple levels, including (1) the *system level*, *e.g.*, where applications can be deployed/removed end-to-end to/from the system, (2) the *application structure level*, *e.g.*, where components (or assemblies of components) associated with one or more applications executing in the system can be added, modified, and/or removed, (3) the *resource level*, *e.g.*, where resources can be made available to application components to ensure their timely completion, and (4) the *application parameter level*, *e.g.*, where configurable parameters (if any) of application components can be tuned. These adaptation levels are interrelated since they directly or indirectly impact system resource utilization and end-to-end QoS, which affects mission success. Adaptations at various levels must, therefore, be performed in a stable and *coordinated* fashion.

To address the unresolved autonomy and adaptive resource

management needs of open DRE systems, we have developed the *Integrated Planning, Allocation, and Control* (IPAC) framework. IPAC integrates decision-theoretic planners, allocators that perform dynamic resource allocation using bin-packing techniques, and controllers that perform runtime system adaptations using control-theoretic techniques.

In our prior work, we developed the *Spreading Activation Partial Order Planner* (SA-POP) [8] and the *Resource Allocation and Control Engine* (RACE) [9]. SA-POP performs decision-theoretic task planning [10], in which uncertainty in task outcomes and utilities assigned to goals are used to determine appropriate sequences of tasks, while respecting resource constraints in DRE systems. RACE provides a customizable and configurable adaptive resource management framework for DRE systems. It allows the system to adapt efficiently and robustly to fluctuations in utilization of system resources, while maintaining QoS requirements. This resource management adaptation is performed through control at the resource level and application parameter level.

SA-POP enables an open DRE system to adapt dynamically to changes in mission goals, as well as changes and losses in system resources. However, by itself it cannot efficiently handle short-term fluctuations in application resource utilization and resource availability because 1) the computational overhead involved in frequent replanning can be very high, and 2) the repeated changes in generated plans may result in system instability where QoS constraints are not satisfied. On the other hand, although RACE's resource management capabilities provide robustness to small variations, it is not suited to handle instabilities when major changes in environmental conditions, mission goals, or resources occur. Neither SA-POP nor RACE used in isolation, therefore, have sufficient capabilities to manage and ensure efficient and stable functioning of open DRE systems. The potential benefits of integrating the adaptation capabilities of SA-POP and RACE were introduced in [8], and the IPAC framework builds up and explicitly demonstrates the advantages of those integration efforts.

This paper provides several contributions to design and experimental research on autonomic computing. It describes and empirically evaluates how the IPAC framework integrates previous work on planning and adaptive resource management for DRE systems to (1) efficiently handle uncertainties, resource constraints, and multiple interacting goals in dynamic assembly of applications, (2) efficiently allocate system resources to application components, and (3) avoid over-utilizing system resources, thereby ensuring system stability and application QoS requirements are met, even under high load conditions. Our results show that IPAC enables the effective and autonomous operation of open DRE systems by performing adaptations at the various levels in a coordinated fashion and ensures overall system stability and QoS.

The remainder of the paper is organized as follows: Section 2 presents an overview of a representative DRE system—the configurable space mission (CSM) system—and describes the system adaptation challenges associated with the autonomous operation of such open DRE systems; Section 3 describes the architecture of IPAC and qualitatively evaluates how it addresses the challenges identified in Section 2; Section 4 quantitatively evaluates how IPAC can address these system adaptation challenges; Section 5 compares our work on IPAC with related work; and Section 6 presents concluding remarks and lessons learned.

2 Configurable Space Mission Systems

This section presents an overview of configurable space mission (CSM) systems, such as NASA's Magnetospheric Multiscale mission system [11] and the proposed Fractionated Space Mission [2], and uses CSMs as a case study to showcase the challenges of open DRE systems and motivate the need for IPAC to provide coordinated system adaptation in the autonomous operation of such systems.

2.1 CSM System Overview

A CSM system consists of several interacting subsystems (both in-flight and stationary) executing in an open environment. Such systems consist of a spacecraft constellation that maintains a specific formation while orbiting in/over a region of scientific interest. In contrast to conventional space missions that involve a monolithic satellite, CSMs distribute the functional and computational capabilities of a conventional monolithic spacecraft across multiple modules, which interact via high-bandwidth, low-latency, wireless links.

A CSM system must operate with a high degree of autonomy, adapting to (1) dynamic addition and modifications of user-specified mission goals/objectives; (2) fluctuations in input workload, application resource utilization, and resource availability due to variations in environmental conditions; and (3) complete or partial loss of resources such as computational power and wireless network bandwidth.

Applications executing in a CSM system, also referred to as science applications, are responsible for collecting science data, processing and analyzing data, storing or discarding the data, and transmitting the stored data to ground stations for further processing. These applications tend to span the entire spacecraft constellation because the fractionated nature of the spacecraft requires a high degree of coordination to achieve mission goals.

QoS requirements of science applications can occasionally be unsatisfied without compromising mission success. Moreover, science applications in a CSM system are often periodic, allowing the dynamic modification of their execution rates at runtime. Resource consumption by—and QoS of—these science applications are directly proportional to their execution rates, *i.e.*, a science application executing at a higher rate contributes a higher value to the overall system QoS, but also consumes resources at a higher rate.

2.2 Challenges Associated with the Autonomous Operation of a CSM System

Challenge 1: Dynamic addition and modifications of mission goals. An operational CSM system can be initialized with a set of goals related to the primary, on-going science objectives. These goals affect the configuration of applications deployed on the system resources, *e.g.*, computational power, memory, and network bandwidth. During normal operation, science objectives may change dynamically and mission goals

can be dynamically added and/or modified as new information is obtained. In response to dynamic additions/modifications of science goals, a CSM system must (re)plan its operation to assemble/modify one or more end-to-end applications (*i.e.*, a set of interacting, appropriately configured application components) to achieve the modified set of goals under current environmental conditions and resource availability. After one or more applications have been assembled, they will first be allocated system resources and then deployed/initialized atop system resources. Section 3.4.1 describes how IPAC resolves this challenge.

Challenge 2: Adapting to fluctuations in input workload, application resource utilization, and resource availability.

To ensure the stability of open DRE systems, system resource utilization must be kept below specified limits, while accommodating fluctuations in resource availability and demand. On the other hand, significant under-utilization of system resources is also unacceptable, since this can decrease system QoS and increase operational cost. A CSM system must therefore reconfigure application parameters appropriately for these fluctuations (*e.g.*, variations in operational conditions, input workload, and resource availability) to ensure that the utilization of system resources converge to the specified utilization bounds ("set-points"). Autonomous operation of the CSM system requires (1) monitoring of current utilization of system resources, (2) (re)planning for mission goals, considering current environmental conditions and limited resource availability, and (3) timely allocation of system resources to applications that are produced as a result of planning. Section 3.4.2 describes how IPAC resolves this challenge.

Challenge 3: Adapting to complete or partial loss of system resources. In open and uncertain environments, complete or partial loss of system resources—nodes (computational power), network bandwidth, and power—may occur during the mission. The autonomous operation of a CSM system requires adaptation to such failures at runtime, with minimal disruption of the overall mission. Achieving this adaptation requires the ability to optimize overall system expected utility (i.e., the sum of expected utilities of all science applications operating in the system) through prioritizing existing science goals, as well as modifying, removing, and/or redeploying science applications. Consequently, autonomous operation of a CSM system requires (1) monitoring resource liveness, (2) prioritizing mission goals, (3) (re)planning for goals under reduced resource availability, and (4) (re)allocating resources to resulting applications. Section 3.4.3 describes how IPAC resolves this challenge.

3 Integrated Planning, Allocation, and Control (IPAC) Framework

Our integrated planning and adaptive resource management architecture, IPAC, enables self-optimization, self-(re)configuration, and self-organization in open DRE systems by providing decision-theoretic planning, dynamic resource allocation, and runtime system control services. IPAC integrates a planner, resource allocator, a controller, and system monitoring framework, as shown in Figure 1.

As shown in Figure 1, IPAC uses a set of resource monitors to track system resource utilization and periodically update the planner, allocator, and controller with current resource utilization (*e.g.*, processor/memory utilization and battery power). A set of QoS monitors tracks system QoS and periodically updates the planner and the controller with QoS values, such as applications' end-to-end latency and throughput. The planner uses its knowledge of the available components' functional characteristics to dynamically assemble applications (*i.e.*, choose and configure appropriate sets of interacting application components) suitable to current conditions and goals/objectives. During this application assembly, the planner also respects resource constraints and optimizes for overall system expected utility.

IPAC's allocators implement resource allocation algorithms, such as multi-dimensional bin-packing algorithms [4], which allocate various domain resources (such as CPU, memory, and network bandwidth) to application components by determining the mapping of components onto nodes in the system domain. After applications have been deployed, IPAC's controller, which implement control-theoretic adaptive resource management algorithms such as EUCON [12], periodically monitors and fine-tunes application/system parameters/properties, such as execution rate, to achieve efficient use of system resources.

The remainder of this section describes IPAC's key *services*: decision-theoretic planning, dynamic resource allocation, and runtime system control. We also show how IPAC can be applied to open DRE system, such as the CSM system described in Section 2.1, to address the challenges associated with the autonomic operation of CSM systems identified in Section 2.2.

3.1 Online Planning using IPAC

Context. Autonomous DRE systems, such as CSMs, operate in dynamic and uncertain environments where local conditions may change rapidly. These changes in local conditions may significantly influence the efficacy of deployed applications in achieving mission goals. Further, communication with mission control systems may involve significant lag times and/or be intermittent.

Problem. To operate efficiently and effectively in such domains requires incorporating some degree of autonomy that allows the system to self-adapt, self-optimize, and self-configure to dynamic changes in local environmental conditions. Moreover, changing mission goals can most effectively be achieved with self-configuration, *i.e.*, when applications are tailored to achieve specified goals in light of local environmental conditions and probabilistic domain information. In addition to these considerations, effective autonomy for open DRE systems also requires the system to self-optimize its operation for achieving mission goals in the face of significant resource constraints.

Solution: A decision-theoretic planner with resource constraints. The IPAC planner performs dynamic assembly of component-based applications that operate with limited resources in uncertain environments. This planning for application assembly is performed by IPAC in terms of abstract *tasks*, which capture the functionality of one or more actual *components*. The architecture of IPAC's planner is shown in Figure 2.

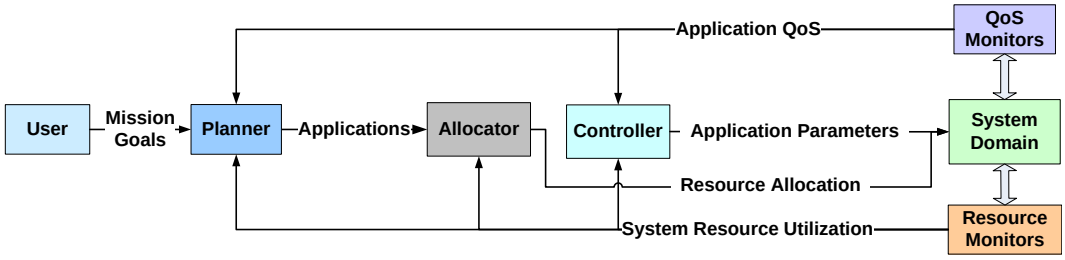


Fig. 1: An Integrated Planning, Resource Allocation, and Control (IPAC) Framework for Open DRE Systems

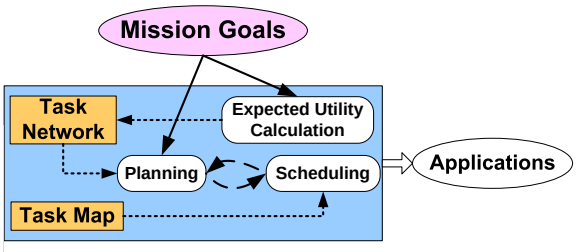


Fig. 2: IPAC Planner Architecture

For the IPAC planner to choose appropriate tasks to achieve a goal, it requires knowledge of preconditions that must be satisfied for each task, its input/output data streams (if any), and the pertinent effects that result from its operation. Uncertainty as to whether tasks will produce the desired output or effects is captured via conditional probabilities associated with the preconditions and effects of a task. Together, these input/output definitions, preconditions/effects, and related conditional probabilities define the *functional signature* of a task. The functional signatures of every task—and consequently all task dependencies—are captured in a *task network* as illustrated in Figure 2. The task network is constructed by domain experts using a domain-specific modeling language in the Generic Modeling Environment (GME) [13].

In addition to knowledge of the functionality provided by the abstract tasks, IPAC's planner translate tasks into appropriately configured application components. Moreover, to ensure applications and their scheduled executions do not violate resource and time constraints, the planner also requires knowledge of a component's *resource signature*, which describes the expected resource consumption and execution time for applicable configurations of the component. To associate each abstract task with a set of concrete components and their individual resource signatures, IPAC uses a *task map* tailored to a specific domain. The task map contains the resource signatures of all components available in the system and is generated by system designers using component profiling.

Given one or more goals specified by a software agent or system user, the IPAC planner uses current conditions and functional knowledge of tasks (from the task network) to generate plans that include data connections and ordering constraints between tasks [8]. The planner uses the probabilistic information from the task network and current conditions to ensure that these plans have a high expected utility (*i.e.*, their probability of successfully achieving provided goals, combined with the utility of those goals, is high compared to other possible plans). During planning, the tasks are also associated with configured components that can implement them and the plan is checked to ensure that overall system resource constraints are not violated [8], [14]. The planner directly translates these plans into assemblies of components, with a schedule of acceptable time windows for their execution. The end product of IPAC's planning process is thus one or more applications assembled from configured components that are likely to achieve the provided goals, given the current local conditions and resource constraints.

3.2 Online Resource Allocation using IPAC

Context. Applications executing in open DRE systems are resource-sensitive (*i.e.*, end-to-end QoS is reduced significantly if the required type and quantity of resources are not provided to the applications at the right time) and require multiple resources, such as memory, CPU, power, and network bandwidth. In these systems resource allocation cannot be performed solely at design-time since system resource availability may vary during run-time. Moreover, input workload affects the utilization of system resources by applications that are already running.

Problem. A key challenge lies in allocating system resources to application components in a timely manner. While many allocation algorithms and heuristics exist, most are applicable only to allocation problems with a single resource. Even among multiple-resource allocation algorithms/heuristics, no single one outperforms all others for finding solutions [15]. Further, the resulting allocations differ depending on the algorithm used, and some allocations may be more desirable than others (*e.g.*, in terms of load-balancing).

Solution: A suite of multi-dimensional bin-packers. IPAC provides an allocator that uses application metadata to allocate domain resources (*e.g.*, CPU and memory) to application components, as shown in Figure 3. The IPAC allocator determines the component-to-node mapping at runtime based on estimated resource requirements of the components, provided in the application metadata, and current node resource availability, provided by resource monitors.

The IPAC allocator uses a suite of allocation heuristics, each a multi-dimensional bin-packing heuristic (*e.g.*, multi-dimensional extensions of best-fit-decreasing and worst-fit-decreasing) with a small running time, to increase the likelihood of finding a valid allocation in a timely manner [15]. These heuristics perform resource allocation by considering

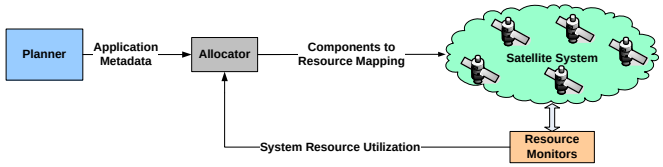


Fig. 3: IPAC's Online Resource Allocation Architecture

each node to be a "bin" with multiple dimensions corresponding to its resources (*e.g.*, CPU and memory) and choosing a bin for each component, into which it is "packed." By executing multiple allocation heuristics, IPAC increases the chances of finding a valid allocation and allows preferential selection among solutions (*e.g.*, choosing the most load-balanced solution).

3.3 Effective System Adaptation using IPAC

Context. In open DRE systems, applications can be added and/or removed at runtime due to dynamically changing mission goals. Moreover, utilization of system resource by applications may be significantly different than their estimated values and availability of system resources may be time-variant. In addition, for applications executing in these systems, the relation between input workload, resource utilization, and QoS cannot be characterized *a priori*. In these systems, failure and/or loss of resources (such as node failure) is not uncommon. To ensure that QoS requirements of applications are met, therefore, open DRE system must be able to *adapt* to dynamically changing events and/or conditions.

Problem. Autonomous operation of open DRE systems require them to adapt, including self-optimize and self-(re)configure, to variations in operational conditions, mission goals, and/or fluctuations in resource availability and demand. As described in Section 1, adaptation in open DRE systems can be performed at various levels, including the *system level*, *application structure level*, *resource level*, and *application parameter level*. As these adaptation decisions are tightly coupled, the key problem lies in ensuring that adaptations at various levels of the system are performed in a stable and *coordinated* fashion.

Solution: Top-down adaptation architecture. IPAC's adaptation architecture is structured in a top-down fashion as shown in Figure 4. IPAC's planner receives feedback on system resource utilization and application QoS from the resource and QoS monitors, respectively, as shown in Figure 4. The planner uses this information to determine when specified goals are not being achieved. In these cases, IPAC's planner performs *coarse-grained* adaptations, such as modifying existing applications (adding, removing, or reconfiguring components) based on current conditions and resource usage.

As shown in Figure 4, after the coarse-grained adaptation decisions have been computed, the planner employs the allocator to compute the allocation of system resources to the newly generated and/or modified application(s). After the allocation is complete, application components are (re)deployed onto the system. The planner updates the controller with the metadata of the newly generated and/or modified application(s).

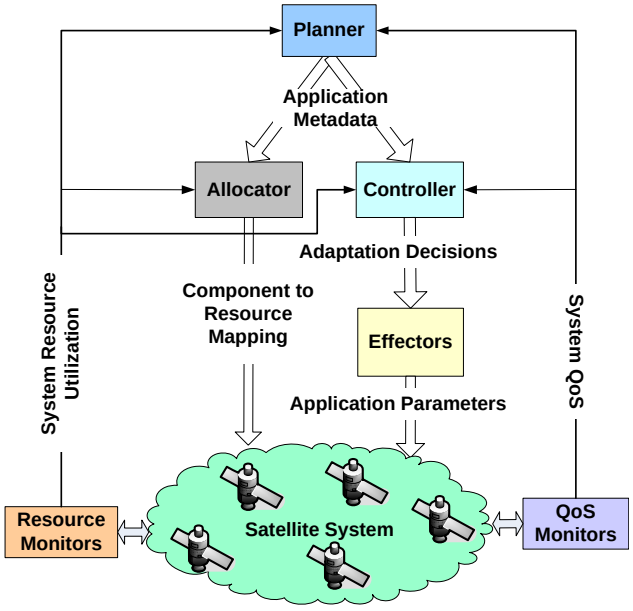


Fig. 4: IPAC's Layered Control Architecture

IPAC's controller implements control-theoretic adaptive resource management algorithms (such as EUCON [12]). It periodically monitors system behavior (resource utilization and QoS) with the aid of the resource and QoS monitors and computes *fine-grained* system adaptation decisions, such as fine-tuning application parameters (*e.g.*, execution rates) and system parameters (operating system and/or middleware QoS parameters). These fine-grained adaptations ensure that system performance and resource utilization requirements are met despite drifts/fluctuations in utilization of system resources and/or application QoS.

Figure 4 also shows how these decisions serve as inputs to IPAC's *effectors*, which modify system parameters (such as execution rates of applications) to achieve controller-recommended adaptation. IPAC's controller and effectors work with its resource monitors and QoS monitors to compensate for drifts/fluctuations in utilization of system resources and/or application QoS. In the current version, IPAC's controller implements the EUCON control algorithm.

The coarse-grained adaptations computed by the planner require longer to implement because they require redeployment of application components. It is therefore preferable to use IPAC's controller to handle fine-grained fluctuations in resource usage and application QoS whenever possible. Although the inputs to both IPAC's planner and the controller include system behavior and performance metrics, the planner uses this information to monitor the evolution of the system with respect to its *long-term* plan/schedule for achieving mission goals and to re-plan/re-schedule when necessary; the controller uses this information to *fine-tune* application/system parameters in response to drifts/fluctuations in utilization of system resources and/or application QoS.

3.4 Addressing CSM System Challenges Using IPAC

We now describe how the capabilities offered by IPAC address the system management challenges for open DRE systems

identified in Section 2.2.

3.4.1 Addressing Challenge 1: Dynamic Addition and Modification of Mission Goals

When IPAC's planner receives a mission goal from a user it assembles an application capable of achieving the provided goal, given current local conditions and resource availability. After the planner assembles an appropriate application, the allocator allocates resources to application components and employs the underlying middleware to deploy and configure the application.

After the application is deployed successfully, the planner updates the controller with the application's metadata including application structure, mapping of allocation components to system resources, and minimum and maximum execution rates. The controller uses this information to dynamically modify system/application parameters (such as execution rates of applications) to accommodate the new application in the system and ensure resources are not over-utilized as a result of this addition. Section 4.4 empirically evaluates the extent to which IPAC's planning, resource allocation, and runtime system adaptation services can improve system performance in when mission goals are dynamically added to the system or modifications to goals deployed earlier are performed.

3.4.2 Addressing Challenge 2: Adapting to Fluctuations in Input Workload and Application Resource Utilization

IPAC tracks system performance and resource utilization via its resource and QoS monitors. IPAC's controller and effectors periodically compute system adaptation decisions and modify system parameters, respectively, to handle minor variations in system resource utilization and performance due to fluctuations in resource availability, input workload, and operational conditions. Section 4.5 empirically validates how IPAC's controller enables the DRE system to adapt to fluctuations in input workload and application resource utilization.

3.4.3 Addressing Challenge 3: Adapting to Complete or Partial Loss of System Resources

When IPAC's controller and effectors cannot compensate for changes in resource availability, input workload, and operational conditions (*e.g.*, due to drastic changes in system operating conditions like complete loss of a node), re-planning in the planner is triggered. The planner performs iterative plan repair to modify existing applications to achieve mission goals. Although this re-planning may result in lower expected utility of some applications, it allows the system to optimize overall system expected utility, even in cases of significant resource loss. Section 4.6 empirically evaluates the extent to which IPAC enables open DRE systems to adapt to loss of system resources.

4 Performance Results and Analysis

This section describes experiments and analyzes results that empirically evaluate the performance of our prototype of the configurable space mission (CSM) case study described in Section 2. These experiments evaluate the extent to which IPAC performs effective end-to-end adaptation, thereby enabling the autonomous operation of open DRE systems. To evaluate how individual services, planning and resource management services, offered by IPAC impact the performance of the system, we ran the experiments in several configurations, *e.g.*, (1) using IPAC with the full set of services (decision-theoretic planning, dynamic resource allocation, and runtime system control services) enabled and (2) with limited sets of IPAC services enabled.

4.1 Hardware and Software Testbed

Our experiments were performed on the ISISLab testbed at Vanderbilt University (www.dre.vanderbilt.edu/ISISlab), which is a cluster consisting of 56 IBM blades powered by Emulab software (www.emulab.net). Each blade node contains 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 [16] was used on all nodes.

We used five blade nodes for the experiments, each acting as a spacecraft in our prototype CSM system. Our middleware platform was CIAO 0.5.10, which is an open-source QoS-enabled component middleware that implements the OMG Lightweight CORBA Component Model (CCM) [17] and Deployment and Configuration [18] specifications. IPAC and the test applications implementing in our CSM system prototype were written in C++ using the CIAO APIs.

4.2 Prototype CSM System Implementation

Mission goals of our prototype CSM system included (1) weather monitoring, (2) monitoring earth's plasma activity, (3) tracking a specific star pattern, and (4) high-fidelity imaging of start constellations. The relative importance of these goals are summarized in Table 1.

#	Goal	Importance
1	Weather Monitoring	100
2	Sunspot Activity Monitoring	80
3	Star Tracking	20
4	Hi-fi Terrestrial Imaging	40

TABLE 1: Utility of Mission Goals

Applications that achieved these goals were periodic (*i.e.*, applications contained a timer component that periodically triggered the collection, filtration, and analysis of science data) and the execution rate of these applications could be modified at runtime. Table 2 summarizes the number of lines of C++ code of various entities in our CIAO middleware, IPAC framework, and prototype implementation of the CSM DRE system case study, which were measured using SLOCCount (www.dwheeler.com/sloccount).

Entity	Total Lines of Source Code
CSM DRE system prototype	18,574
IPAC framework	80,253
CIAO middleware	511,378

TABLE 2: Lines of Source Code for Various System Elements

4.3 Experiment Design

As described in Section 2, a CSM system is subjected to (1) dynamic addition of goals and end-to-end applications, (2) fluctuations in application workload, and (3) significant changes in resource availability. To validate our claim that IPAC enables the autonomous operation of open DRE systems, such as the CSM system, by performing effective end-to-end adaptation, we evaluated performance of our prototype CSM system performance when (1) goals were added at runtime, (2) application workloads were varied at runtime, and (3) a significant drop in available resources occurred due to node failure.

To evaluate the improvement in system performance due to IPAC, we initially indented to compare the system behavior (system resource utilization and QoS) with and without IPAC. However, without IPAC, a planner, a resource allocator, and a controller were not available to the system. Therefore, dynamic assembly of applications that satisfy goals, runtime resource allocation to application components, and online system adaptation to variations in operating conditions, input workload, and resource availability were not possible. In other words, without IPAC our CSM system would reduce to a "static-system" that cannot operate autonomously in open environments.

To evaluate the performance IPAC empirically, we structured our experiments as follows:

- **Experiment 1** presented in Section 4.4 compares the performance of the system that is subjected to dynamic addition of user goals at runtime when the full set of services (*i.e.*, planning, resource allocation, and runtime control) offered by IPAC are employed to manage the system versus when only the planning and resource allocation services are available to the system.
- **Experiment 2** presented in Section 4.5 compares the performance of the system that is subjected to fluctuations input workload when the full set of services offered by IPAC are employed to manage the system versus when only planning and resource allocation services are available to the system.
- **Experiment 3** presented in Section 4.6 compares the performance of the system that is subjected to node failures when the full set of services offered by IPAC are employed to manage the system versus when only resource allocation and control services are available to the system.

For all the experiments, IPAC's planner was configured to use overall system expected utility optimization and respect total system CPU usage constraints. Likewise, the allocator was configured to use a suite of bin-packing algorithms with worst-fit-decreasing and best-fit-decreasing heuristics. Finally, the controller was configured to employ the EUCON control algorithm to compute system adaptation decisions.

4.4 Experiment 1: Addition of Goals at Runtime

4.4.1 Experiment Design

This experiment compares the performance of the system when the full set of services (*i.e.* planning, resource allocation, and runtime control) offered by IPAC are employed to manage the system versus when only the planning and resource allocation services are available to the system. This experiment also adds user goals dynamically at runtime. The objective is to demonstrate the need for—and empirically evaluate the advantages of—a specialized controller in the IPAC architecture. We use the following metrics to compare the performance of the system under the different service configurations:

1) **System downtime,** which is defined as the duration for which applications in the system are unable to execute due to resource reallocation and/or application redeployment.
2) **Average application throughput,** which is defined as the throughput of applications executing in the system averaged over the entire duration of the experiment.
3) **System resource utilization,** which is measure of the processor utilization on each node in the system domain.

We demonstrate that a specialized controller, such as EUCON, enables the system to adapt more efficiently to fluctuations in system configuration, such as addition of applications to the system. In particular, we empirically show how the service provided by a controller is complementary to the services of both the allocator and the planner.

4.4.2 Experiment Configuration

During system initialization, time $T = 0$, the first goal (weather monitoring) was provided to the planner by the user, for which the planner assembled five applications (each with between two and five components). Later, at time $T = 200sec$, the second goal (monitoring earth's plasma activity) goal was provided to the planner, which assembled two applications (with three to four components each) to achieve this goal. Next, at time $T = 400sec$, the third goal (start tracking) was provided to the planner, which assembled one application (with two components) to achieve this goal. Finally, at time $T = 600sec$, the fourth goal (hi-fi imaging) was provided to the planner, which assembled an application with four components to achieve this goal. Table 3 summarizes the provided goals—and the applications deployed corresponding to these goals—as a function of time. Table 4 summarizes the application configuration, *i.e.*, minimum and maximum execution rates, estimated average resource utilization of components that make up each application, and the ratio of estimated resource utilization between the worst case workload and the average case workload.

Time (sec)	Goal	Application #
0 - 200	Weather Monitoring	1 - 5
200 - 400	Sunspot Activity Monitoring	6 - 7
400 - 600	Star Tracking	8
600 - 800	Hi-fi Terrestrial Imaging	9

TABLE 3: Set of Goals and Corresponding Applications as a Function of Time

For this experiment, the sampling period of the controller was set to 2 seconds. The processor utilization set-point of the controller, as well as the *bin-size*, of each node was selected to be 0.7, which is slightly lower than RMS [19]

Application	Exec. Rate (Hz)			Net Estimated	Component Average Resource Util.					Util. Ratio
	Min	Max	Init.	Resource Util.	1	2	3	4	5	Average Case : Worst Case
1	15	155	60	0.3	0.15	0.1	0.05	0	0	1 : 1.86
2	35	165	85	0.1	0.05	0.05	0	0	0	1 : 3.00
3	10	140	50	0.5	0.2	0.1	0.1	0.05	0.05	1 : 1.22
4	30	170	80	0.3	0.25	0.05	0	0	0	1 : 3.00
5	35	180	90	0.45	0.2	0.1	0.1	0.05	0	1 : 1.22
6	10	140	65	0.35	0.15	0.1	0.05	0.05	0	1 : 3.00
7	35	170	95	0.35	0.25	0.05	0.05	0	0	1 : 1.86
8	60	95	80	0.35	0.3	0.05	0	0	0	1 : 1.86
9	40	85	60	0.40	0.15	0.10	0.10	0.5	0	1 : 1.20

TABLE 4: Application Configuration

utilization bound of 0.77. IPAC allocator was configured to use the standard best-fit-decreasing and worst-fit-decreasing bin-packing heuristics.

4.4.3 Analysis of Experiment Results

When IPAC featured the planner, the allocator, and the controller, allocation was performed by the allocator using the average case utilization values due to the availability of the controller to handle workload increases that would result in greater than average resource utilization. When IPAC featured only the planner and the allocator, however, all allocations were computed using the worst case resource utilization values (use of average case utilizations can not be justified because workload increases would overload the system without a controller to perform runtime adaptation). Tables 5 and 6 summarize the initial allocation of components to nodes (for applications 1 - 5 at time $T = 0$ corresponding to the weather monitoring goal), as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

Node	Estimated Utilization	Items (Application, Component)			
1	0.35	(4,1)	(2,1)	(3,5)	
2	0.35	(3,1)	(5,2)	(4,2)	
3	0.35	(5,1)	(5,3)	(5,4)	
4	0.30	(1,1)	(3,3)	(2,2)	
5	0.30	(1,2)	(3,2)	(1,3)	(3,4)

TABLE 5: Allocation of Applications 1 - 5 using Average Case Utilization

Node	Estimated Utilization	Items (Application, Component)			
1	0.43	(4,1)	(3,5)		
2	0.40	(3,1)	(5,3)	(1,3)	
3	0.39	(5,1)	(5,2)	(3,4)	
4	0.44	(1,1)	(3,3)	(2,2)	(5,4)
5	0.40	(1,2)	(3,2)	(2,1)	(4,2)

TABLE 6: Allocation of Applications 1 - 5 using Wost Case Utilization

At time $T = 200sec$, when the applications for the plasma activity monitoring goal were deployed (applications 6 and 7 as specified in Table 4), the system reacted differently when operated with the controller than without it. With the controller, enough available resources were expected (using average case utilization values), so the allocator could incrementally allocate applications 6 and 7 in the system thus required no reallocation or redeployment.

In contrast, when the system operated without the controller, a reallocation was necessary as an incremental addition of applications 6 and 7 to the system was not possible (allocations were based on worst case utilization values). The reallocation of resources requires redeployment of application components and, therefore, increases system/application downtime. Tables 7 and 8 summarize the revised allocation of components to nodes (for applications 1 - 7), as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

Node	Estimated Utilization	Items (Application, Component)				
1	0.45	(4,1)	(2,1)	(3,5)	(6,2)	
2	0.45	(3,1)	(5,2)	(4,2)	(6,3)	(7,2)
3	0.45	(5,1)	(5,3)	(5,4)	(6,4)	(7,3)
4	0.55	(1,1)	(3,3)	(2,2)	(7,1)	
5	0.45	(1,2)	(3,2)	(1,3)	(3,4)	(6,1

TABLE 7: Allocation of Applications 1 - 7 using Average Case Utilization

Node	Estimated Utilization	Items (Application, Component)				
1	0.615	(4,1)	(5,3)	(6,4)	(5,4)	
2	0.575	(7,1)	(3,2)	(2,2)	(7,2)	
3	0.605	(6,1)	(1,2)	(3,3)	(4,2)	(7,3)
4	0.610	(3,1)	(1,1)	(2,1)	(1,3)	(3,4)
5	0.610	(5,1)	(6,2)	(5,2)	(6,3)	(3,5)

TABLE 8: Allocation of Applications 1 - 7 using Wost Case Utilization

At time $T = 400sec$, when the application corresponding to the star tracking goal was provided (application 8), resources were insufficient to incrementally allocate it to the system, both with and without the controller, so reallocation was necessary.

When the IPAC was configured without the controller, the allocator was unable to find a feasible allocation using the best-fit decreasing heuristic. However, IPAC's allocator was able to find a feasible allocation using the best-fit decreasing heuristic. Tables 10 and 11 summarize the allocation of components to nodes, as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

At time $T = 600sec$, application corresponding to the hi-fi imaging goal (application 9) had to be deployed. When operating without the controller, it was not possible to find any allocation of all nine applications, and the system continued to operate with only the previous eight applications. In contrast, when the system included the controller, average case utilization values were used during resource allocation,

and application 9 was incrementally allocated and deployed in the system.

When the system was operated with the full set of services offered by IPAC the overall system downtime[1] due to resource reallocation and application redeployment was 8534.375 ms compared to 15613.162 ms when the system was operated without the system adaptation service of IPAC. It is clear that the system downtime is significantly (50%) lower when the system was operating with the full set of services offered by IPAC than when the system was operating without the controller.

From Figure 5, it is clear that system resources are significantly underutilized when operating without the controller but are near the set-point when the controller is used. Underutilization of system resources results in reduced QoS, which is evident from Table 9, showing the overall system QoS.[2]

Application	Average Throughput (Hz)	
	With the Controller	Without the Controller
1	149.973	59.871
2	159.236	84.802
3	100.700	49.624
4	116.453	79.814
5	175.156	89.653
6	25.076	63.212
7	37.370	94.876
8	89.620	79.894
9	40.514	N/A
Entire System	99.344	66.860

TABLE 9: Experiment 1: Comparison of System QoS

4.4.4 Summary

This experiment compared system performance under dynamic addition of mission goals when the full set of IPAC services (*i.e.*, planning, resource allocation, and runtime control) were employed to manage the system versus when only the planning and resource allocation services were available. Significant difference in system evolution were observed due to the fact that when the system was operated without the controller, resources were reallocated more often than when the controller was available. Higher system downtime resulted, further lowering average throughput and resource utilization. Moreover, when the system was operated with the controller, additional mission goals could be achieved by the system, thereby improving the overall system utility and QoS.

From these results, it is clear that without the controller, even dynamic resource allocation is inefficient due to the necessary pessimism in component utilization values (worst case values from profiling). Lack of a controller thus results in (1) under-utilization of system resources, (2) low system QoS, and (3) high system downtime. In contrast, when IPAC featured the planner, the allocator, and the controller, resource allocation was significantly more efficient. This efficiency stemmed from the presence of the controller, which ensures system resources are not over-utilized despite workload increases. These results also demonstrate that when IPAC operated with a full set of services it enables the efficient and autonomous operation of the system despite runtime addition of goals.

1. To measure the system downtime, we repeated the experiment over 100 iterations and computed the average system downtime.

2. In this system, overall QoS is defined as the total throughput for all active applications.

4.5 Experiment 2: Varying Input Workload

4.5.1 Experiment Design

This experiment executes an application corresponding to the weather monitoring, monitoring earth's plasma activity, and star tracking goals (applications 1 - 8 described in Table 4), where the input workload is varied at runtime. This experiment demonstrates the adaptive resource management capabilities of IPAC under varying input workload. We compare the performance of the system when the full set of services offered by IPAC (*i.e.*, planning, resource allocation, and runtime control) are employed to manage the system versus when only planning and resource allocation services are available to the system. 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.

4.5.2 Experiment Configuration

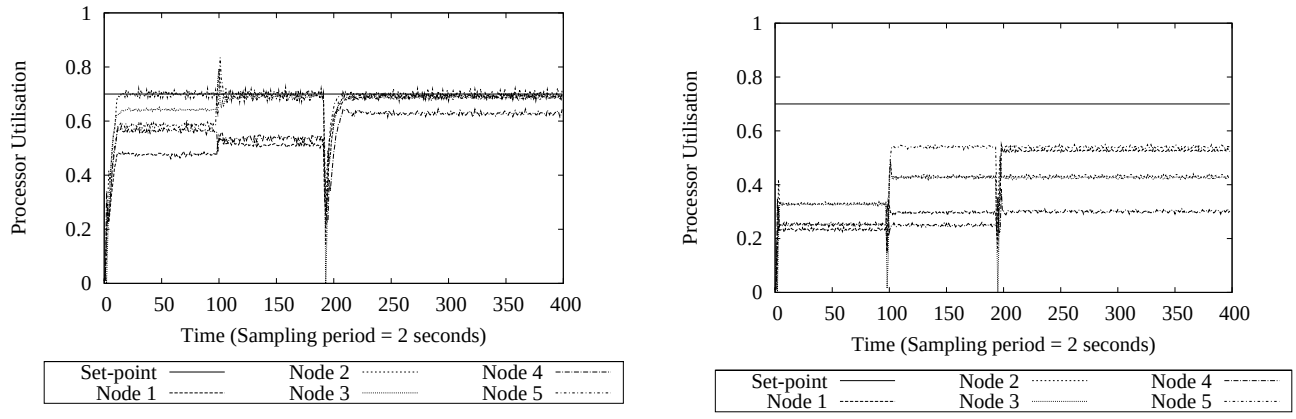
At time $T = 0$, the system was initialized with applications 1 - 8 as specified in Table 4. Upon initialization, applications execute at their initialization rate specified in Table 4. When IPAC featured the planner, the allocator, and the controller, allocation was performed by the allocator using the average case utilization values due to the availability of the controller to handle workload increases that would result in greater than average resource utilization. When IPAC featured only the planner and the allocator, however, all allocations were computed using the worst case resource utilization values. Tables 10 and 11 summarize the allocation of components to nodes, as well as the estimated resource utilization, using average case and worst case utilization values, respectively.

Node	Estimated Utilization	Items (Application, Component)				
1	0.55	(8,1)	(3,3)	(2,1)	(3,4)	(6,4)
2	0.55	(4,1)	(1,2)	(5,2)	(3,5)	(7,2)
3	0.55	(7,1)	(3,2)	(5,3)	(4,2)	(7,3)
4	0.55	(3,1)	(1,1)	(6,2)	(5,4)	(8,2)
5	0.50	(5,1)	(6,1)	(1,3)	(2,2)	(6,3)

TABLE 10: Allocation of Applications 1 - 8 using Average Case Utilization

Each applicationss 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 application/components meet their deadlines/sub-deadlines if the schedulable utilization bound of RMS [19] 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 RMS utilization bound.



(a) Utilization with the Controller

(b) Utilization without the Controller

Fig. 5: Experiment 1: Comparison of Processor Utilization

Node	Estimated Utilization	Items (Application, Component)									
1	0.69	(8,1)	(6,1)	(2,1)							
2	0.70	(4,1)	(7,1)								
3	0.70	(3,1)	(5,1)	(1,1)	(1,3)						
4	0.685	(6,2)	(1,2)	(3,2)	(3,3)	(5,2)	(2,2)				
5	0.695	(5,3)	(4,2)	(6,3)	(6,4)	(7,2)	(7,3)	(8,2)	(3,4)	(3,5)	(5,4)

TABLE 11: Allocation of Applications 1 - 8 using Wost Case Utilization

Table 12 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.

Time (sec)	Input Workload
0 - 150	Low
150 - 450	Medium
450 - 600	High
600 - 900	Medium
900 - 1,000	Low

TABLE 12: Input Workload as a Function of Time

4.5.3 Analysis of Experiment Results

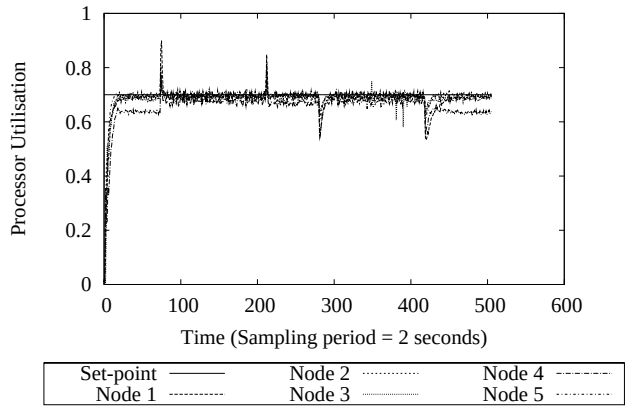
When the IPAC controller is available to the system it dynamically modifies the execution rates of applications within the bounds $[min, max]$ specified in Table 4 to ensure that the resource utilization on each node converges to the specified set-point of 0.7, despite fluctuations in input workload. When IPAC is *not* configured with the controller (*i.e.*, only the planner and the allocator are available), however, applications execute at their initialization rate specified in Table 4.

Figure 6a, Figure 7a, and Table 12 show the execution of the system when it contains the IPAC controller. During $0 \leq T \leq 150$, 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 IPAC does not provide the controller service,however, Figures 6b and 7b show that the applications execute at a constant rate (initialization rate) and system resources are severely underutilized.

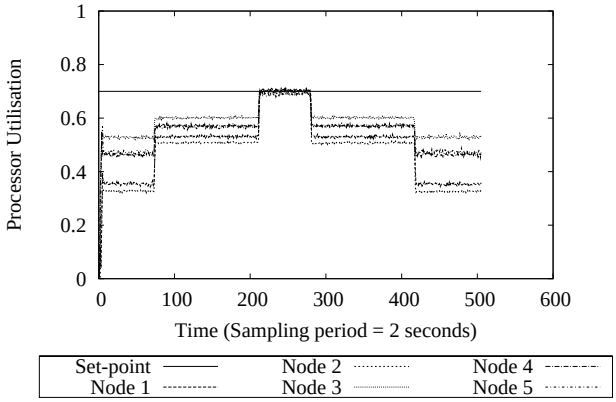
When input workload is increased from low to medium, at $T = 150s$, the corresponding increase in the processor utilization can be seen in Figure 6. Figures 6a and 7a show that when IPAC included the controller, although the processor utilization increased above the set-point, within a few sampling periods the controller restored the processor utilization to the desired set-point of 0.7 by dynamically reducing the execution rates of applications. Under both service configuration of IPAC, with the controller and without the controller, the deadline miss ratio was 0 throughout the duration of the experiment. Figure 6a shows that the application deadline miss ratio was unaffected by the short duration during which processor utilization was above the set-point. Finally, Figure 6b shows that without the controller, the system resources remained under-utilized even after the workload increase.

At $T = 450s$, 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 6. Figures 6a and 7b show that the controller was again able to dynamically modify the application execution rates to ensure that the utilization converged to the desired set-point. Figure 6b shows that when IPAC did not feature the controller, the processor utilization was at the set-point under high workload conditions (corresponding to the worst case resource utilization used to determine the allocation of components to processors in that case).

At $T = 600s$, when the input workload was reduced from high to medium, from Figure 6 it can be seen that the processor utilization on all the nodes decreased. When IPAC included the controller, however, the controller restored the processor

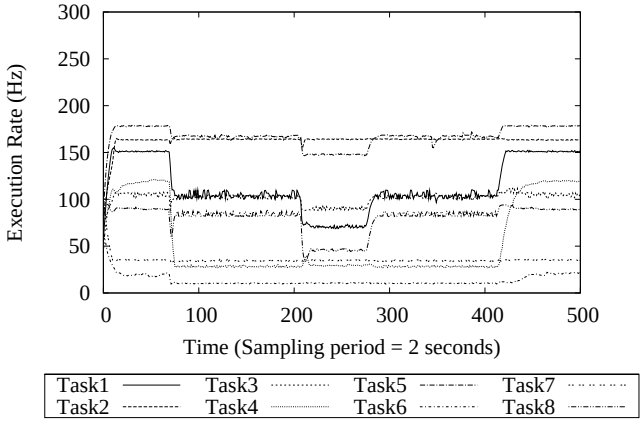


(a) With the Controller

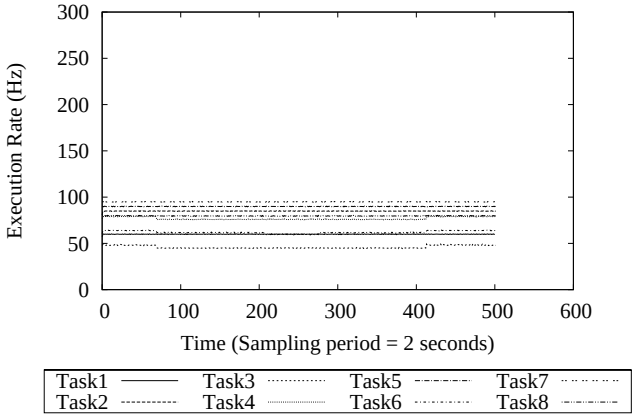


(b) Without the Controller

Fig. 6: Experiment 2: Comparison of Processor Utilization



(a) With the Controller



(b) Without the Controller

Fig. 7: Experiment 2: Comparison of Application Execution Rates

utilization to the desired set-point of 0.7 within a few sampling periods. Without the controller, processor utilization remained significantly lower than the set-point. Similarly, at $T = 900s$, the input workload was further reduced from medium to low, and Figure 6 shows another decrease in processor utilization across all nodes. When IPAC featured the controller, processor utilization again returned to the desired set-point within a few sampling periods. Without the controller, processor utilization remained even further below the set-point.

Figure 6 shows that system resources are significantly underutilized when operating without the controller, but are near the set-point when the controller is used. Underutilization of system resources results in reduced QoS, which is evident from Table 13, showing the overall system QoS.[3] In contrast, when IPAC featured the controller, 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 the Controller	Without the Controller
1	113.17	59.930
2	162.817	84.903
3	101.240	45.964
4	54.507	76.909
5	166.959	89.905
6	13.460	62.088
7	35.219	94.896
8	80.019	79.702
Entire System	90.923	74.287

TABLE 13: Experiment 2: Comparison of System QoS

4.5.4 Summary

This experiment compared system performance during input workload fluctuations when the system was operated with the full set of IPAC services (*i.e.* planning, resource allocation, and runtime control) versus when only the planning and resource allocation services were available to the system. The results

3. In this system, overall QoS is defined as the total throughput for all active applications.

show how IPAC and its controller (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.

4.6 Experiment 3: Varying Resource Availability

4.6.1 Experiment Design

This experiment demonstrate the need for—and advantages of—a planner in our IPAC architecture. It also demonstrates that although a specialized controller can efficiently handle minor fluctuations in the system, it is unable to handle major fluctuations in the system, such as loss of one or more nodes in the system.

We compare the performance of the system when the full set of services offered by IPAC (*i.e.*, planning, resource allocation, and runtime control) are employed to manage the system versus when only resource allocation and control services are available to the system. We use system expected utility and system resource utilization as metrics to empirically compare the performance of the system under each service configuration.

4.6.2 Experiment Configuration

For this experiment, the goals provided to the system were (1) weather monitoring, (2) sunspot monitoring, (3) star-tracking, and (4) hi-fi imaging goals. The sampling period of the controller was set to be 2 seconds. The processor utilization set-point of the controller, as well as the *bin-size*, of each node was selected to be 0.7. Under both configurations of IPAC (*i.e.*, (1) when IPAC featured the planner, allocator, and controller and (2) when IPAC featured only the allocator and the controller), allocation was performed by the allocator using the average case utilization values due to the availability of the controller to handle workload increases that would result in greater than average resource utilization.

When IPAC featured only the allocator and the controller, the allocator is augmented such that if it is unable to allocate all applications given the reduced system resources, the allocator incrementally removes applications from consideration by lowest *utility density* until a valid allocation can be found. We define utility density as the expected utility of the application divided by its expected resource usage.

4.6.3 Analysis of Experiment Results

When IPAC featured only the allocator and the controller, the complete loss of a node triggered reallocation by the allocator. With the reduced system resource, however, the allocator was able to allocate applications corresponding to the weather monitoring, plasma monitoring, and hi-fi imaging goals only.

In contrast, when IPAC featured the planner, the allocator, and the controller, the complete loss of a node triggers re-planning in the planner. The planner then assembled a new set of applications, taking into account the significant reduction in system resources. Although some applications had a lower expected utility than the original ones, all four goals were still achieved with the resources of the four remaining nodes.

Figure 8 shows that both with and without the planner, the controller ensures that the resource utilization on all the nodes are maintained within the specified bounds. Table 14 compares the utility of the system when IPAC did/did-not feature the planner. This figure shows how system adaptations performed by the planner in response to failure of a node result in higher system utility compared to the system adaptation performed by just the allocator and the controller. The results Table 14

Application	Expected Utility	
	With Planner	Without Planner
1	18	18
2	6	6
3	30	30
4	20	20
5	26	26
6	38	40
7	36	40
8	16	–
9	40	40
Entire System	230	220

TABLE 14: Experiment 3: Comparison of System Utility

occur because IPAC's planner was able to assemble modified applications for some mission goals (corresponding to applications 6, 7, and 8), albeit with somewhat lower expected utility, whereas the allocator had to completely remove an application to meet the reduced resource availability.

4.6.4 Summary

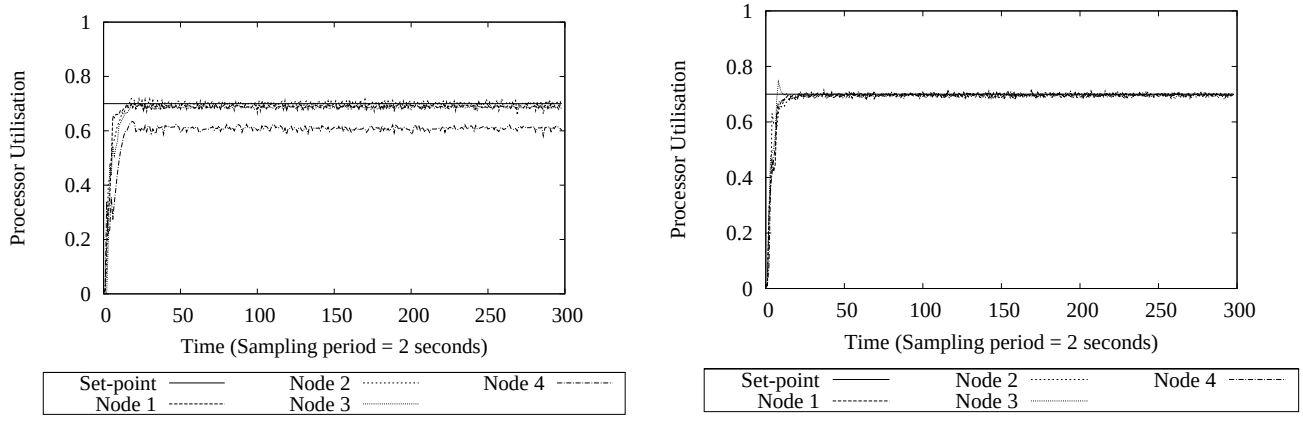
This experiment shows that although a specialized controller can efficiently handle minor fluctuations in resource availability, it may be incapable of effective system adaptation in the case of major fluctuations, such as loss of one or more nodes in the system. Even with the addition of an intelligent resource allocation scheme, system performance and utility may suffer unnecessarily during major fluctuations in resource availability. In contrast, IPAC's planner has knowledge of system component *functionality* and desired mission goals. As a result, it can perform more effective system adaptation in the face of major fluctuations, such as the loss of a system node.

5 Related Work

The overview of autonomic computing presented in [3] identifies various aspects a self-managed autonomic computing system, which includes self-organization, self-reconfiguration, self-optimization, and self-protection. Of these aspects, our research on IPAC focuses on self-organization, self-reconfiguration, and self-optimization of open DRE systems.

The work in [20] describes a utility-driven self-adaptive middleware that processes numerous data-streams simultaneously. This research presents a self-adaptation algorithm that scales efficiently as the number of data-screams increases. The objective of this research is to maximize the utility of the system, despite changing in (1) network and resource conditions and (2) business/user policies.

The work in [21] employs utility functions to efficiently manage, as well as continually optimize, the use of system



(a) With the Planner

(b) Without the Planner

Fig. 8: Experiment 3: Comparison of Processor Utilization

resources in data-centers. In this research, resources are dynamically (re)allocated to applications such that the system objective, which is specified using utility functions, is maximized throughout the lifespan of the system.

AGILE [22] is as a policy expression language and an integration framework for various autonomic computing techniques and technologies. AGILE facilitates the configuration of run-time adaptation of autonomic systems using policy based mechanisms. AGILE, as an integration framework, enables the seamless integration and interoperability of the different system adaptation and management technologies, such as signal processing, automated trend analysis and utility functions, within a single system.

Our approach to system optimization, adaptation, and management using IPAC is similar to the utility based system management and optimization research presented in [20] and [21]. IPAC employs a combination of decision-theoretic planning and control-theoretic resource management, however, to efficiently manage and adapt the system to variations in operational conditions, mission goals, and fluctuations in resource availability and/or demand. Moreover, IPAC performs system adaptation as various levels in a coordinated fashion to effectively adapt the system to these chances. In summary, IPAC's novelty stems from its integration of (1) online decision-theoretic planning, (2) online resource allocation, and (3) runtime system adaptations.

There has been significant work in recent years on planning in dynamic and uncertain environments, such as spacecraft missions. The Remote Agent used on NASA's Deep Space One mission includes autonomous, goal-directed planning under resource constraints [23]. IPAC's planner goes beyond that of the Remote Agent by composing and configuring component-based applications and explicitly modeling uncertainty in the environment to optimize expected utility of plans. Like IPAC, many other planners extend classical notions of planning to include uncertainty. In particular, some planners allow uncertainty about both environment and action outcome (*e.g.*, C-SHOP [24] using hierarchical planning and Drips [25], which produce contingent plans), as in IPAC. While these planners generate a series of actions to be executed, IPAC's planner also provides the capability to select and configure software components, dynamically composing applications for deployment.

The composition/configuration of application components is related to work on service composition in service oriented architectures and web services, such as Synthy [26], which produces contingent plans for combining and deploying web services. Synthy performs a similar function to that of IPAC's planner in that it composes web services from components and represents their functional and non-functional (*e.g.*, QoS) properties separately. However, Synthy requires user input during the composition process to efficiently minimize relevant contingencies, while IPAC's planner is designed to perform autonomously, making choices based on expected utility, rather than producing contingency plans. Further, IPAC's planner is distinguished from service composition planners like Synthy by its focus on resource-constrained environments through incorporation of resource constraints in the planning process.

IPAC integrates three forms of system adaptation and management, namely decision-theoretic planning, runtime resource allocation, and control-theoretic resource management techniques. IPAC can benefit from integration frameworks, such as AGILE [22], with the integration of other autonomic computing techniques and technologies and thereby provide the system with self-protection and self-healing capabilities, in addition to the self-organization, self-reconfiguration, and self-optimization capabilities it currently provides.

6 Concluding Remarks

Autonomous operation of open DRE systems requires robust adaptation of system functionality to current goals and conditions, as well as the enforcement of end-to-end application resources and QoS requirements. Open DRE systems run in environments where operational conditions, input workload, and resource availability are subject to dynamic changes. To meet end-to-end QoS in these dynamic environments, open

DRE systems can benefit from an integrated planning resource allocation, and control frameworks that monitors system resources, performs efficient planning, application workload management, and enables efficient resource provisioning for executing applications.

This paper described the *Integrated Planning, Allocation, and Control* (IPAC) framework, which is our integrated planning and adaptive resource management framework that provides planning, end-to-end adaptation and resource management for open DRE systems. IPAC enables open DRE systems to operate autonomously by (1) dynamic assembly of high expected utility applications, (2) monitoring of system resource utilization and application QoS, (3) performing fine-grained adaptations in response to drifts/fluctuations in utilization of system resources and/or application QoS, and (4) performing functional system adaptation in response to changing environmental conditions or significant variation in system resource availability.

As described in Section 1, adaptations in open DRE systems can be performed at various levels and adaptation decisions must be performed in a coordinated fashion. To ensure adaptations performed by IPAC do not jeopardize system stability, IPAC's control architecture has a top-down structure, *i.e.*, the system adaptation layer, implemented by the planner, reacts to events and the application adaptation layer, implemented by the controller, reacts to minor drifts/fluctuations in system behavior. This design is relatively conservative, *e.g.*, in certain cases the adaptations performed by the application adaptation layer might be suboptimal compared to the adaptations that could potentially be performed by the system adaptation layer. The benefits of IPAC's design, however, is that it ensures system stability, reduces system down-time, and ensures system adaptations at various levels are coordinated.

The experimental results presented in this paper demonstrate the following benefits of using the full set of services offered by IPAC to manage open DRE systems: (1) system resources are efficiently utilized, (2) more goals/applications can be accommodated by the system, (3) system downtime is reduced, (4) utilization of system resources is maintained within a specified set-point, and (5) system QoS is improved significantly.

IPAC is an open-source software that can be obtained along with our middleware distribution from http://download.dre.vanderbilt.edu/.

REFERENCES

[1] D. Suri, A. Howell, N. Shankaran, J. Kinnebrew, W. Otte, D. C. Schmidt, and G. Biswas, "Onboard Processing using the Adaptive Network Architecture," in *Proceedings of the Sixth Annual NASA Earth Science Technology Conference*, College Park, MD, June 2006.

[2] 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, Apr. 2006.

[3] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[4] J. W. S. Liu, *Real-time Systems*. New Jersey: Prentice Hall, 2000.

[5] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, and L. Abeni, "Adaptive Reservations in a Linux Environment," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 238–245.

[6] T. F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," *IEEE: Control Systems*, vol. 23, no. 3, pp. 74–90, June 2003.

[7] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Syst.*, vol. 23, no. 1-2, pp. 85–126, 2002.

[8] 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.

[9] N. Shankaran, D. C. Schmidt, X. D. Koutsoukos, Y. Chen, and C. Lu, "Design and Performance Evaluation of Resource-Management Framework for End-to-End Adaptation of Distributed Real-time Embedded Systems," *Journal on Embedded Systems: Special issue on Operating System Support for Embedded Real-Time Applications*, 2008.

[10] S. Bagchi, G. Biswas, and K. Kawamura, "Task Planning under Uncertainty using a Spreading Activation Network," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 30, no. 6, pp. 639–650, Nov. 2000.

[11] S. Curtis, "The Magnetospheric Multiscale Mission...Resolving Fundamental Processes in Space Plasmas," *NASA STI/Recon Technical Report N*, pp. 48 257–+, Dec. 1999.

[12] 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.

[13] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-Integrated Development of Embedded Software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 145–164, Jan. 2003.

[14] P. Laborie, "Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results." *Artif. Intell.*, vol. 143, no. 2, pp. 151–188, 2003.

[15] N. Roy, J. S. Kinnebrew, N. Shankaran, G. Biswas, and D. C. Schmidt, "Toward Effective Multi-capacity Resource Allocation in Distributed Real-time and Embedded Systems," in *Proceedings of the 11th International Symposium on Object/Component/Service-oriented Real-time Distributed Computing)*. Orlando, Florida: IEEE, May 2008.

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

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

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

[19] 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.

[20] V. Kumar, B. F. Cooper, and K. Schwan, "Distributed Stream Management using Utility-Driven Self-Adaptive Middleware," *icac*, vol. 0, pp. 3–14, 2005.

[21] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility Functions in Autonomic Systems," *icac*, vol. 00, pp. 70–77, 2004.

[22] R. J. Anthony, "Policy-Centric Integration and Dynamic Composition of Autonomic Computing Techniques," in *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*. Washington, DC, USA: IEEE Computer Society, 2007, p. 2.

[23] N. Muscettola, P. Nayak, B. Pell, and B. Williams, "Remote Agent: to boldly go where no AI system has gone before," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 5–47, 1998.

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

[25] 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.

[26] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava, "Synthy: A system for end to end composition of web services," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 4, pp. 311–339, 2005.